21世纪软件工程专业规划教材

软件测试

周元哲 编著

10010101000101
111010010010
10010101000101
111010010010
1000101
10010101000101
111010010010

清华大学出版社

21世纪软件工程专业规划教材

软件测试

周元哲 编著

清华大学出版社
北京

内 容 简 介

本书较为全面、系统地涵盖了当前业界测试领域的理论和实践知识，反映了当前最新的软件测试理论、标准、技术和工具，展望了软件测试的发展趋势。

全书共分三大部分，分别是测试理论、测试实践和测试考试指导。第一部分内容主要包括软件测试概论、软件测试基本知识、软件测试过程、黑盒测试、白盒测试、自动测试技术、性能测试、面向对象测试、嵌入式测试和软件测试管理。第二部分内容主要包括软件测试工具、测试管理工具、性能测试工具、缺陷管理工具、单元测试工具、功能测试工具、嵌入式测试工具等。第三部分内容主要包括四级软件测试工程师考试、企业招聘测试工程师考试和微软公司测试的一些情况。

本书适合作为高等院校相关专业软件测试的教材或教学参考书，也可以供从事计算机应用开发的各类技术人员应用参考，或用作全国计算机软件测评师考试、软件技术资格与水平考试的培训资料。

图书在版编目(CIP)数据

软件测试 / 周元哲编著. --北京：清华大学出版社，2013
21世纪软件工程专业规划教材
ISBN 978-7-302-33192-6

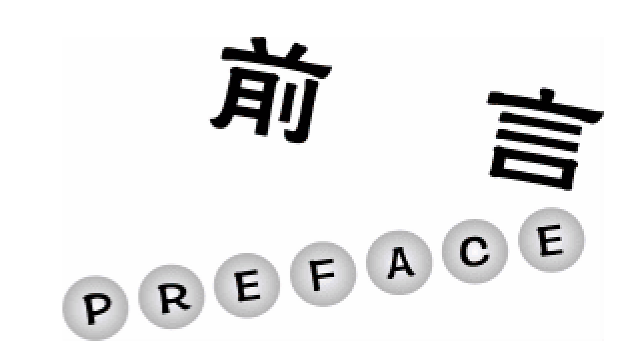
Ⅰ. ①软…　Ⅱ. ①周…　Ⅲ. ①软件－测试－高等学校－教材　Ⅳ. ①TP311.5

中国版本图书馆 CIP 数据核字(2013)第 159103 号

责任编辑：张　玥　顾　冰
封面设计：傅瑞学
责任校对：时翠兰
责任印制：刘海龙

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn，010-62795954
印 装 者：清华大学印刷厂
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**17　　**字　　数：**387 千字
版　　次：2013 年 9 月第 1 版　　**印　　次：**2015 年 7 月第 3 次印刷
印　　数：4001～5500
定　　价：33.00 元

产品编号：053822-01



随着软件的规模和复杂性的大幅度提升，如何保证软件质量的可靠性变得日益突出。软件测试作为成为保证软件质量的关键技术之一，同时也是软件开发过程中的一个重要环节，其理论知识和技术工具都在不断革新。

本书较为全面地涵盖了当前业界测试领域的专业知识，追溯了软件测试的发展史，反映了当前最新的软件测试理论、标准、技术和工具，展望了软件测试的发展趋势。全书共分三大部分，分别是测试理论、测试实践和测试考试指导。第一部分内容主要包括软件测试概论、软件测试基本知识、软件测试流程、黑盒测试、白盒测试、性能测试、面向对象测试、嵌入式测试和软件质量保证。第二部分内容主要包括软件测试自动化与测试工具、测试管理工具、性能测试工具、缺陷管理工具、单元测试工具、功能测试工具和嵌入式测试工具等。第三部分内容主要包括四级软件测试工程师考试、企业招聘测试工程师考试和微软公司软件测试的一些情况。

本书在写作过程中参阅了大量中外文的专著、教材、论文、报告及网上的资料，由于篇幅所限，未能一一列出。在此，向各位作者表示敬意和衷心的感谢。

软件测试从软件工程中演化而来，并且还在不断地发展之中。在学习本书之前，需要一些先行知识作为本书的支撑，如掌握一门高级语言，这包括 Visual Basic 或 Java 语言、数据库、数据结构以及软件工程的基本理论知识等。

西安邮电学院计算机科学与技术学院胡滨、潘晓英、刘海、刘有耀对本书的编写给予了大力的支持，并提出了指导性意见。北京化工大学信息科学与技术学院计算机系系主任赵瑞莲教授、清华大学出版社对本书的写作大纲、写作风格等提出了很多宝贵的意见。

本书介绍了软件测试的基本理论和当前流行的一些软件测试工具的应用，内容精练、文字简洁、结构合理、综合性强，明确定位面向初、中级读者，由“入门”起步，侧重“提高”。特别适合作为高等院校相关专业软件测试的教材或教学参考书，也可以供从事计算机应用开发的各类技术人员应用参考，或用作全国计算机软件测评师考试、软件技术资格与水平考试的培训资料。

由于作者水平有限，时间紧迫，本书难免有不足之处，诚恳期待读者的批评指正，以使本书日臻完善。我们的电子信箱是 zhouyuanzhe@163.com。

编 者
2013 年 6 月

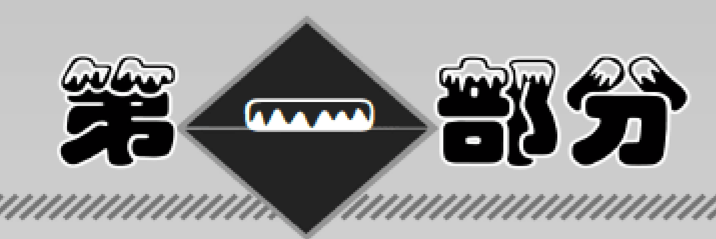
目 录

CONTENTS

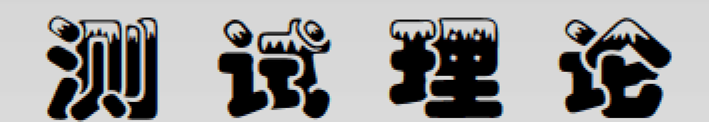
第一部分　测试理论

第二部分　测 试 实 践

第三部分　测试考试指导



第一部分



测试理论



软件测试概论

本章介绍软件的发展历史、软件项目的特性和生命周期等内容,并对软件缺陷的内容以及缺陷的后果做了介绍,引出了软件测试的内容,为学习本书后续内容作必要准备。

1.1 软件与软件项目

软件是一系列按照特定顺序组织的计算机数据和指令的集合。一般来讲软件被划分为系统软件、应用软件和介于这两者之间的中间件。其中系统软件为计算机使用提供最基本的功能,但是并不针对某一特定应用领域。而应用软件则恰好相反,不同的应用软件根据用户和所服务的领域提供不同的功能。

一般认为,软件包括如下内容:

(1) 运行时,能够提供所要求功能和性能的指令或计算机程序集合。

(2) 程序能够满意地处理信息的数据结构。

(3) 描述程序功能需求以及程序如何操作和使用所要求的文档。

1.1.1 软件发展史

20 世纪 50 年代初期至 60 年代中期是软件发展的第一阶段(又称为程序设计阶段)。此时硬件已经通用化,而软件的生产却是个体化。软件产品为专用软件,规模较小,功能单一,开发者即使用者,软件只有程序,无文档。软件设计在人们的头脑中完成,形成了“软件等于程序”的错误观念。

第二阶段从 20 世纪 60 年代中期至 70 年代末期是程序系统阶段。此时多道程序设计技术、多用户系统、人机交互式技术、实时系统和第一代数据库管理系统的出现,出现了专门从事软件开发的“软件作坊”,软件广泛应用,但软件技术和管理水平相对落后,导致“软件危机”出现。软件危机主要表现在以下几个方面:

(1) 软件项目无法按期完成,超出经费预算,软件质量难以控制;

(2) 开发人员和开发过程之间管理不规范,约定不严密,文档书写不完整,使得软件维护费用高,某些系统甚至无法进行修改;

(3) 缺乏严密有效的质量检测手段,交付给用户的软件质量差,在运行中出现许多问题,甚至带来严重的后果;

(4) 系统更新换代难度大。

第三阶段称为软件工程阶段，从 20 世纪 70 年代中期至 80 年代中期，由于微处理器的出现、分布式系统广泛应用，以软件的产品化、系列化、工程化和标准化为特征的软件产业发展起来，软件开发有了可以遵循的软件工程化的设计准则、方法和标准。1968 年，北大西洋公约组织的计算机科学家在联邦德国召开国际会议，讨论软件危机问题，正式提出并使用“软件工程”概念，标志软件工程诞生。软件工程学涉及与生产软件相关的所有活动，包括计算机科学、管理学、经济学、心理学等，其研究的主要内容是：如何应用科学的理论和工程上的技术来指导软件的开发，从而达到以较少的投资获得高质量软件的最终目标。

第四阶段是从 20 世纪 80 年代中期至今，客户端/服务器(C/S)体系结构、特别是 Web 技术和网络分布式对象技术飞速发展，导致软件体系结构向更加灵活的多层分布式结构演变，CORBA、EJB、COM/DCOM 等三大分布式的对象模型技术相继出现。2006 年，面向服务架构(Service-Oriented Architecture，SOA)作为下一代软件架构，作为“抽象、松散耦合和粗粒度”的软件架构，根据需求通过网络对松散耦合的粗粒度应用组件进行分布式部署、组合和使用，主要用于解决传统对象模型中无法解决的异构和耦合问题。

至此，软件发展经历了 Mainframe 结构、Client/Server 结构、B/S 多层分布式结构、SOA 的演变过程，整个软件系统变得越来越分散、越来越开放、越来越强调互操作性。

1.1.2 项目特性

项目是为完成某项独特的产品、服务或成果所做的临时性努力。项目具有以下特征：

(1) 项目具有明确的目标。项目的目标就是完成某一产品、服务或预期成果，而且在定义项目目标时通常带有进度和成本的限制。例如，某一项目的目标是：“在 6 个月内，以 2 万元的成本开发完成学校网络教学平台”。

(2) 项目具有临时性。临时性是指每一个项目都有开始和结束时间。当项目的目标已经达到，或由于各种原因项目不需要再持续下去时，项目即达到了它的终点，项目团队也会解散。任何项目的期限都是有限的，项目不是持续不断的努力。

(3) 项目具有独特性，也称一次性。不同于那种重复性的日常工作，项目创造独特的产品、服务或成果。例如，设计和建造“国家歌剧院”是一个项目，而每天的卫生保洁工作不是项目。

(4) 项目是逐步完善(渐进明细)的。逐步完善意味着分步、连续的积累和逐步的细化。在项目初期，对项目范围、规模、成本、进度的估计和计划都是粗粒度的，随着项目的进展，对这些因素的理解会逐渐地深入和细化。

(5) 项目使用的资源是受到限制的。资源包括人员、设备、材料等，可供一个项目使用的这些资源是有限的。

(6) 项目具有一定程度的不确定性。在一个项目开始时，通常要对项目的进度、成本等进行估计，并据此提出项目目标，制定项目计划。但在项目执行过程中，人员、资金、技术、市场等因素在不断变化，项目可能会遇到各种各样的风险，这会给项目带来一定程度的不确定性，使项目不能完全按照原有计划执行，项目目标也不可能完全达到。

项目作为一种特殊的活动，有效地利用各种资源，通过执行一系列相互联系的任务而

达到一个独特的目标。项目普遍存在于人类社会中，以下是项目的一些例子：

- 开发一个新的产品。
- 设计和实现一个新版的计算机应用系统。
- 一个工厂的现代化改造。
- 建造一座建筑。
- 自然灾害后一个城市的重建。
- 某软件企业的 CMMI3 级认证。
- 举行一次学术研讨会。
- 举办一个一百周年庆典。

1.1.3 软件项目

软件项目是一种特殊的项目，其特殊性表现在它的目标是生产软件产品。软件产品与其他类型的项目产品有很大的差异，Fred Brooks 在他的文章《没有银弹》中，总结了软件的以下特点。

(1) 复杂性。软件实体可能比任何人类创造的其他实体都复杂。软件系统有数量极大的状态，这使得设计、描述和测试软件系统都非常困难；软件中没有任何两个部分是完全相同的，软件系统的扩展也不是相同元素的重复添加，而是不同元素实体的添加，大多数情况下，这些元素之间的交互途径以非线性递增的方式增长，因此整个软件系统的复杂度以更大的非线性级数增长。

(2) 可变性。由于软件是纯粹的逻辑思维的产物，它可以很容易地被改变，可以无限地扩展。而实际上软件也总是处于持续的变更之中，用户需求的改变，运行环境和硬件平台的改变都会强迫软件随之变化。

(3) 不可见性。软件是逻辑实体，不具有空间的形体特征，因此是不可见的和无法可视化的。用图形描述软件会受到很大限制，一种图形只能描述软件某一部分或某一方面的属性，而不能全面形象地描述软件。这种不可见性不仅给软件设计带来困难，也严重阻碍了人员之间的交流。

正由于软件具有以上特点，软件产品的生产比一般产品的生产更难于控制。因此软件项目虽然具有项目的一般特性，但它是一个新的领域，具有以下特点。

(1) 知识密集型，技术含量高。软件项目是知识密集型项目，技术性很强，需要大量高强度的脑力劳动。项目工作十分细致、复杂和容易出错。软件项目不需要使用大量的物质资源，而主要是使用人力资源，因此人员的因素极为重要，项目团队成员的结构、技能、责任心和团队精神对软件项目的成功与否有着决定性的影响。

(2) 涉及多个专业领域，多种技术综合应用。软件项目属于典型的跨学科合作项目，例如开发大型管理信息系统就需要项目成员具有行业的业务知识、数据库技术、程序设计技术和信息安全技术等多专业领域知识。

(3) 项目范围和目标的灵活性。随着项目的进展，客户需求可能会发生变化，从而导致项目范围和目标的变化。软件开发不像其他产品的生产，有着非常具体的标准和检验方法，软件的标准柔性很大，衡量软件是否成功的重要标准就是用户满意度，但用户满意

度这个标准在软件开发前很难精确地、完整地表达出来。

(4)风险大,收益大。由于技术的高度复杂性和需求等因素的不确定性,软件项目风险控制难度较大,项目的成功率较低,但是一旦某个软件产品获得成功,将会带来相对高额的回报。

(5)客户化程度高。项目的独特性在软件领域表现得更为突出,不同的软件项目之间差别较大。软件开发商往往要根据客户的具体要求提供独特的解决方案,即使有现成的解决方案,通常也需要进行一定的客户化工作。

(6)过程管理的重要性。软件项目需要对整个项目过程进行严格和科学的管理,尤其是对大型、复杂的软件项目。“质量产生于过程”,必须监控软件开发的过程和中间结果。没有严格的过程管理,开发人员的个人能力再强也没有用。

目前,软件项目的开发和运作远远没有其他领域的项目规范,很多的理论还不能适应所有的软件项目,经验在软件项目中仍起很大的作用。

1.1.4 项目生命周期

在实施项目管理的过程中,将项目划分为便于进行管理的一系列阶段,并在不同的阶段执行一系列具体的项目管理活动。项目生命周期的长度根据项目的不同有很大差别,可能只有几个星期,也可能长达数年。在项目生命周期的各个阶段,人力、资源和费用的投入是不平均的,项目开始时投入的资源等比较低,然后逐渐升高,在项目的实施阶段,达到最高峰,此后逐渐下降,直到项目的终止。

软件项目的生命周期通常可划分为4个阶段:识别需求、方案设计、项目执行和项目收尾,如图1.1所示。

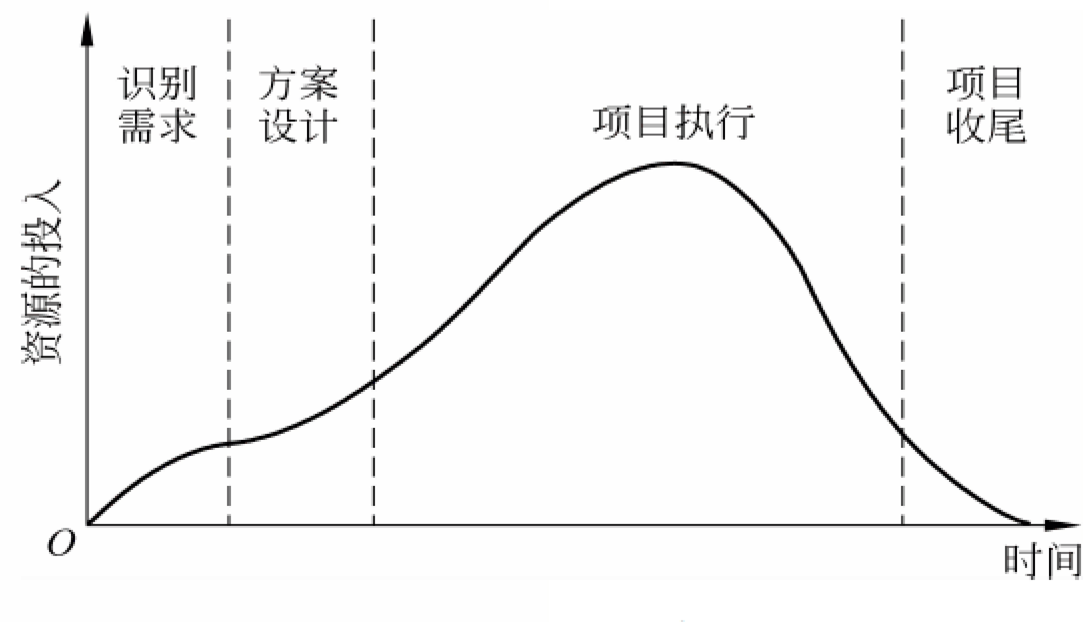


图1.1 项目的生命周期

1. “识别需求”阶段

由客户提出需求,项目组与客户商议后确定需解决的问题。这里的“客户”是指为完成项目而提供资金的人或组织。这些被客户识别出的需求或问题通常被记录下来,提交给相关的人员或组织去解决,客户在提出需求时往往会附加时间和成本限制。

2."方案设计"阶段

根据客户的需求提出一个解决方案。首先要进行可行性研究,对新系统可能的开发和运行成本及其效益进行估计。如果可行性研究的结果表明预期的项目可行,就要设计解决方案以及与解决方案相配套的进度、成本、质量、风险、人力资源等方面的规划。可能会有多个组织向客户提交他们的解决方案,客户经过比较,选择其中一个,然后双方签订合同。这就是通常的项目招标方式。在有些情况下,客户也会选择其内部团队来完成项目,而不承包给外部组织。

3."项目执行"阶段

项目执行是根据解决方案,实施项目活动,满足客户的需求,在该阶段的末尾通常需要对项目产品或服务进行验证。在这一阶段还要不断监控项目的执行过程,测量项目的实际进程和质量指标是否与计划一致,如果测量结果表明出现了偏差,要立即采取纠正措施,以使项目恢复到正常轨道,或者更正计划的不合理之处。

4."项目收尾"阶段

项目最后一个阶段的任务是执行项目的收尾工作,例如确认所有的项目可交付物都已移交给客户,所有的费用都已经清算。对项目承担者来说,收尾阶段的一个重要任务是对项目过程进行总结,得到对本组织的改进有益的经验教训。项目组需要调查客户的满意度,收集客户和项目团队成员的建议,从而能够改进以后项目的性能。

需要注意的是,项目的生命周期与项目产品的生命周期是两个不同的概念,一个项目结束后,项目产品或服务的生命周期通常不会结束。对于一个软件项目来说,当把软件产品移交并通过用户验收后,通常项目就结束了,但软件产品还有很长的使用和维护期,在此期间对于比较大的软件修改维护任务,可另外设立项目来进行管理。

1.2 软件缺陷

1.2.1 软件缺陷案例

1963年,由于用FORTRAN程序设计语言编写的飞行控制软件中的循环语句"DO 5 I=1,3"误写为"DO 5 I=1.3",DO语句少了一个逗号,结果导致美国首次金星探测飞行失败,造成价值约1000多万美元的损失。

1979年,新西兰航空公司的一架客机因计算机控制的自动飞行系统发生故障而撞在阿尔卑斯山上,机上257名乘客全部遇难。

1992年10月26日,伦敦救护中心的计算机辅助发送系统刚启动就崩溃了,导致这个全世界最大的每天要接运五千多病人的救护机构全部瘫痪。

1994年,美国迪士尼公司的《狮子王》软件在少数系统中能正常工作,但在常见系统

中不行。后来证实，迪士尼公司没有对市场上投入使用的各种 PC 机型进行正确的测试。也就在同年，Intel 奔腾浮点除法发生软件缺陷，Intel 为处理软件缺陷支付 4 亿多美元。

1996 年 6 月 4 日，耗资 80 亿美元的欧洲航空航天局发射的阿里亚娜 501 火箭，发射升空 37 秒后爆炸。原因是主发动机打火顺序开始 37 秒后，制导信息由于惯性制导系统的软件出现规格和设计错误而完全遗失。

1999 年 9 月，火星气象人造卫星在经过 41 周 4.16 亿英里飞行后，在即将要进入火星轨道时失败了，为此，美国投资 5 万美元调查事故原因，发现太空科学家洛克希德·马丁采用的是英制（磅）加速度数据，而喷气推进实验室则采用的是公制（牛顿）加速度数据进行计算。此事故的发生就是因为集成测试的失败所致。

临近 2000 年时，计算机业界一片恐慌，这就是著名的“千年虫”问题。其原因是在 20 世纪 70 年代，由于计算机硬件资源很珍贵，程序员为节约内存资源和硬盘空间，在存储日期数据时，只保留年份的后 2 位，如“1980”被存储为“80”。当 2000 年到来时，问题出现了，计算机无法分清“00”是指“2000 年”还是“1000 年”。例如银行存款的软件在计算利息时，本应该用现在的日期“2000 年 1 月 1 日”减去当时存款的日期。但是，由于“千年虫”的问题，结果用“2000 年 1 月 1 日”减去当时存款的日期，存款年数就变为负数，导致顾客反要付给银行支付巨额的利息。为了解决“千年虫”问题，花费了大量的人力、物力和财力。

2003 年 8 月 14 日下午 4 时 10 分，美国及加拿大部分地区发生历史上最大的停电事故。15 日晚逐步恢复。造成的经济损失在 250 亿到 300 亿美元之间。原因分析：俄亥俄州的第一能源（FirstEnergy）公司下属的电力监测与控制管理系统软件 XA/21 出现错误，系统中重要的预警部分出现严重故障，负责预警服务的主服务器与备份服务器连接失控，错误没有得到及时通报和处理，最终多个重要设备出现故障，导致大规模停电。

2005 年 4 月 20 日上午 10 时 56 分，中国银联系统通信网络和主机出现故障，造成辖内跨行交易全部中断。这是 2002 年中国银联成立以来，首次全国性因系统故障造成的跨行交易全面瘫痪。原因是银联新近准备上线的某外围设备的隐性缺陷诱发了跨行交易系统主机的缺陷，使主机发生故障。

2007 年 8 月 14 日 14 时，美国洛杉矶国际机场电脑发生故障，60 个航班的 2 万旅客无法入关。直至次日凌晨 3 时 50 分，所有滞留旅客才全部入关。原因分析：包含旅客姓名和犯罪记录的部分数据系统（海关和边境保护系统：决定旅客是否可以进入美国领土）瘫痪 2004 年 9 月发生过类似问题

2008 年，我国举行了首次奥运会。10 月 30 日上午 9 时北京奥运会门票面向境内公众销售第二阶段正式启动，系统访问流量猛增，官方票务网站流量瞬时达到每小时 800 万次，超过了系统设计每小时 100 万次的承受量，奥运门票系统访问量超计划 8 倍，造成网络拥堵，售票速度慢或暂时不能登录系统的情况，直接造成公众无法及时提交购票申请，官方票务系统已于前天下午 6 点关闭，北京奥运票务中心就此向广大境内公众购票人发布了致歉信……

通常认为，符合下面 4 个规则之一就是软件缺陷。

（1）软件未达到软件规格说明书中规定的功能。

（2）软件出现了产品说明书中指明不会出现的错误。

（3）软件功能超出了产品说明书中指明的范围。

（4）软件测试人员认为软件难于理解，不易使用，运行速度慢，或者最终用户认为软件使用效果不好。

【例 1-1】 软件缺陷举例。

计算器说明书一般声称该计算器将准确无误地进行加、减、乘、除运算。如果测试人员选定了数值，按下＋号后，再按下一数值，如果没有任何结果出现，或者得到了错误的答案，根据第一条规则，这是一个缺陷。

计算器产品说明书指明计算器不会出现崩溃、死锁或者停止反应等情况，而测试人员按键后，计算器却停止接收等，根据第二条规则，这是一个缺陷。在测试过程中发现，由于电池没电等原因可以导致计算不正确，但产品说明书上没有指出在此情况下应该如何进行处理，这也是一个缺陷。

若在进行测试时，发现除了产品说明书规定的加、减、乘、除运算功能之外，还能够进行求平方根的运算，而这一功能并没有在说明书中给出，根据第三条规则，这是一个缺陷。

如果测试人员发现计算器某些功能不好使用，如按键太小、显示屏在亮光下无法看清等，根据第四条规则，这是一个缺陷。

1.2.2 软件缺陷

缺陷产生由软件本身特点、软件项目管理和团队工作等 3 个方面的原因导致而成。

1. 软件开发过程自身的特点造成

首先，由于软件本身固有的复杂性。当前软件系统具有图形用户界面、B/S 结构、面向对象设计、分布式运算、底层通信协议、超大型关系型数据库等多种模块，从而使得软件系统的复杂性呈指数增长。其次，由于需求变化增加了软件系统开发的复杂性，产生了大量不确定因素，导致许多缺陷产生。

2. 软件项目管理的问题

首先，软件项目开发过程中往往由于时间的限制，导致缺少文档的编写，而文档的贫乏容易使得代码维护和修改变得很难。其次，由于开发流程不够完善，存在较多的随机性和缺乏严谨的内审或评审机制，容易产生问题。

3. 团队工作的问题

由于团队组成人员本身由于个人的认知层面、各自拥有的知识、处事原则各不相同，交流不充分等，使得误解难免会产生。

软件缺陷内容包括缺陷标识、缺陷类型、缺陷严重程度、缺陷产生可能性、缺陷优先级、缺陷状态、缺陷起源、缺陷来源、缺陷原因。

(1) 缺陷标识:是标记某个缺陷的唯一的表示,可以使用数字序号表示。

(2) 缺陷类型:是根据缺陷的自然属性划分缺陷种类,如表 1.1 所示。

表 1.1　软件缺陷类型列表

缺陷类型	描　　述
功能	影响了各种系统功能、逻辑的缺陷
用户界面	影响了用户界面、人机交互特性,包括屏幕格式、用户输入灵活性、结果输出格式等方面的缺陷
文档	影响发布和维护,包括注释、用户手册、设计文档
软件包	由于软件配置库、变更管理或版本控制引起的错误
性能	不满足系统可测量的属性值,如执行时间、事务处理速率等
系统/模块接口	与其他组件、模块或设备驱动程序、调用参数、控制块或参数列表等不匹配、冲突

(3) 缺陷严重程度:是指因缺陷引起的故障对软件产品的影响程度,所谓“严重性”是指在测试条件下,一个错误在系统中的绝对影响,如表 1.2 所示。

表 1.2　软件缺陷严重等级列表

缺陷严重等级	描　　述
致命	系统任何一个主要功能完全丧失、用户数据受到破坏、系统崩溃、悬挂、死机或者危及人身安全
严重	系统的主要功能部分丧失、数据不能保存,系统的次要功能完全丧失,系统所提供的功能或服务受到明显的影响
一般	系统的次要功能没有完全实现,但不影响用户的正常使用。例如,提示信息不太准确或用户界面差、操作时间长等一些问题
较小	使操作者不方便或遇到麻烦,但它不影响功能的操作和执行,如个别的不影响产品理解的错别字、文字排列不对齐等一些小问题

(4) 缺陷产生的可能性:指缺陷在产品中发生的可能性,通常可以用频率来表示,如表 1.3 所示。

表 1.3　缺陷产生可能性列表

缺陷产生可能性	描　　述
总是	总是产生这个软件缺陷,其产生的概率是 100%
通常	通常情况下会产生这个软件缺陷,其产生的概率大概是 80%～90%
有时	有的时候产生这个软件缺陷,其产生的概率大概是 30%～50%
很少	很少产生这个软件缺陷,其产生的概率大概是 1%～5%

(5) 缺陷优先级:指缺陷必须被修复的紧急程度。“优先级”的衡量抓住了在严重性中没有考虑的重要程度因素,如表 1.4 所示。

表 1.4　软件缺陷优先级列表

缺陷优先级	描　述
立即解决	缺陷导致系统几乎不能使用或测试不能继续，需立即修复
高优先级	缺陷严重，影响测试，需要优先考虑
正常排队	缺陷需要正常排队等待修复
低优先级	缺陷可以在开发人员有时间的时候被纠正

一般来讲，缺陷严重等级和缺陷优先级相关性很强，但是，同时具有低优先级和高严重性的错误是可能的，反之亦然。例如，产品徽标是重要的，一旦它丢失了，这种缺陷是用户界面的产品缺陷，但是它阻碍产品的形象，成为了优先级很高的软件缺陷。

（6）缺陷状态：指缺陷通过一个跟踪修复过程的进展情况，也就是在软件生命周期中的状态基本定义，如表 1.5 所示。

表 1.5　软件缺陷状态列表

缺陷状态	描　述
激活或打开	问题还没有解决，存在源代码中，确认“提交的缺陷”，等待处理，如新报的缺陷
已修正或修复	已被开发人员检查、修复过的缺陷，通过单元测试，认为已解决但还没有被测试人员验证
关闭或非激活	测试人员验证后，确认缺陷不存在之后的状态
重新打开	测试人员验证后，还依然存在的缺陷，等待开发人员进一步修复
推迟	这个软件缺陷可以在下一个版本中解决
保留	由于技术原因或第三者软件的缺陷，开发人员不能修复的缺陷
不能重现	开发不能复现这个软件缺陷，需要测试人员检查缺陷复现的步骤

（7）缺陷来源：指缺陷所在的地方，如文档、代码等，如表 1.6 所示。

表 1.6　软件缺陷来源列表

缺陷来源	描　述
需求说明书	需求说明书的错误或不清楚引起的问题
设计文档	设计文档描述不准确和需求说明书不一致的问题
系统集成接口	系统各模块参数不匹配、开发组之间缺乏协调引起的缺陷
数据流（库）	由于数据字典、数据库中的错误引起的缺陷
程序代码	纯粹在编码中的问题所引起的缺陷

（8）缺陷根源：指造成上述错误的根本因素，以寻求软件开发流程的改进、管理水平的提高，如表 1.7 所示。

表 1.7　软件缺陷根源列表

缺陷根源	描　　述
测试策略	错误的测试范围，误解了测试目标，超越测试能力等
过程，工具和方法	无效的需求收集过程，过时的风险管理过程，不适用的项目管理方法，没有估算规程，无效的变更控制过程等
团队	项目团队职责交叉，缺乏培训，没有经验等
组织和通信	缺乏用户参与，职责不明确，管理失败等
硬件	硬件配置不对、缺乏，或处理器缺陷导致算术精度丢失，内存溢出等
软件	软件设置不对、缺乏，或操作系统错误导致无法释放资源，工具软件的错误、编译器的错误，千年虫问题等
工作环境	组织机构调整、预算改变、工作环境恶劣，如噪音过大

1.2.3　软件缺陷流程

软件缺陷导致项目进度难于控制，项目管理难度加大。如图 1.2 所示，大量的软件错误往往只有到了项目后期系统测试时才能够被发现，解决问题所花的时间很难预料，经常导致项目进度无法控制，推迟了项目的发布日期。

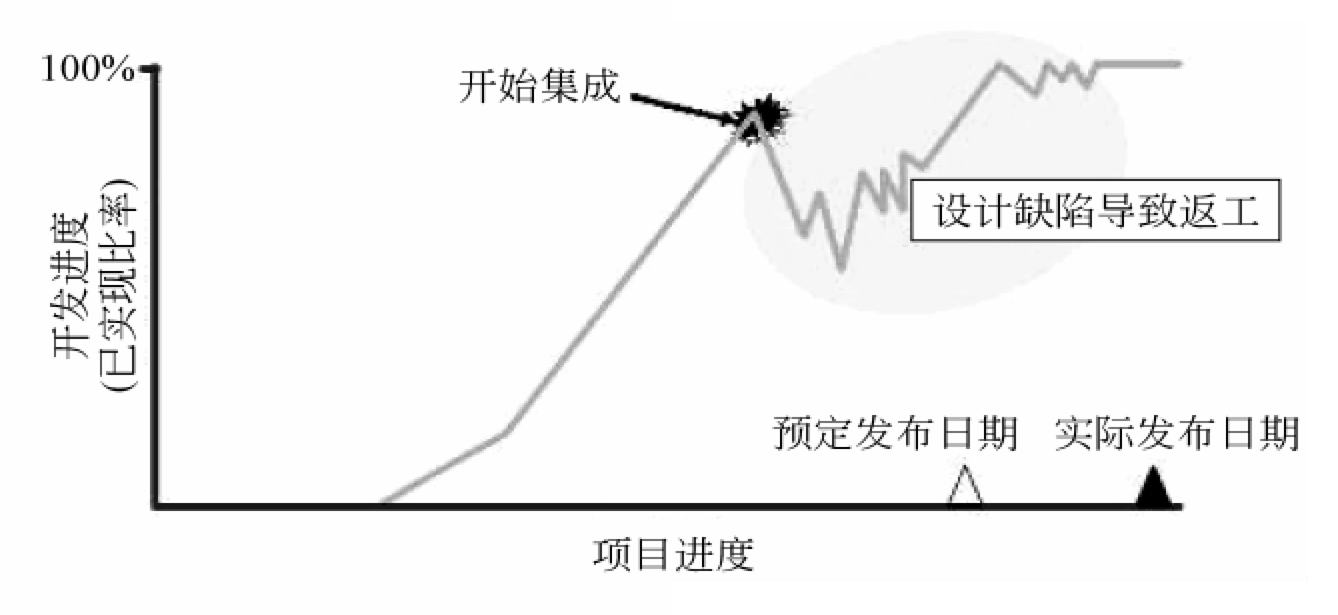


图 1.2　推迟项目的发布日期

缺陷的修复费用随着软件开发阶段的推移将急剧上升，在需求分析阶段发生的缺陷，在产品发布之后修复缺陷的成本将是在软件需求阶段修复缺陷的 100 倍，甚至更高，缺陷的延迟解决必然导致整个项目成本的急剧增加，如图 1.3 所示。

和软件生命周期一样，一个缺陷从它产生到终结的过程，称之为缺陷生命周期，又称缺陷处理流程。每个缺陷都具有 8 种生命状态，即缺陷初始状态、缺陷分配状态、缺陷重新分配状态、缺陷修复状态、缺陷验证状态、缺陷重新打开状态、缺陷关闭状态，如图 1.4 所示。

软件测试是发现和修复软件缺陷的重要办法。作为保证软件质量的一种辅助而且必需的手段，软件测试是根据软件开发各个阶段的规格说明和程序的内部结构设计测试用例，通过测试用例执行程序，发现软件故障的过程，其根本目的是以尽可能少的时间和人力发现并改正软件中潜在的各种故障和缺陷。

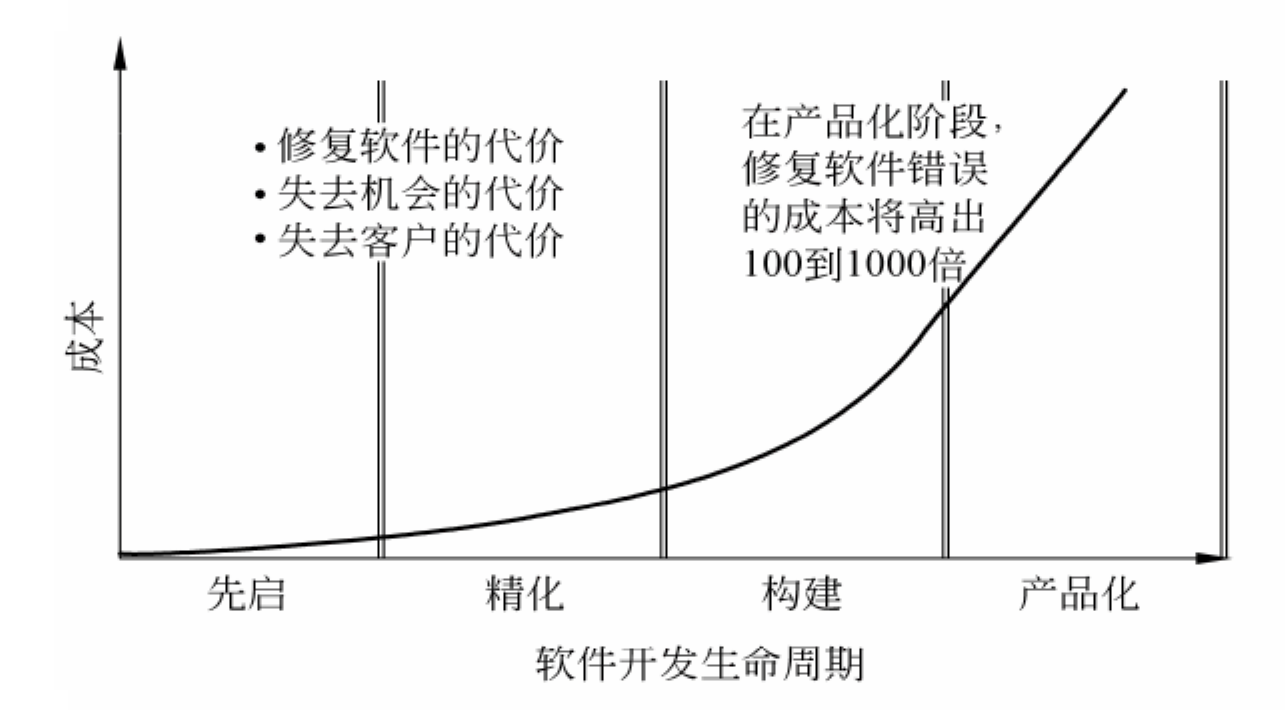


图 1.3 加大软件修复成本

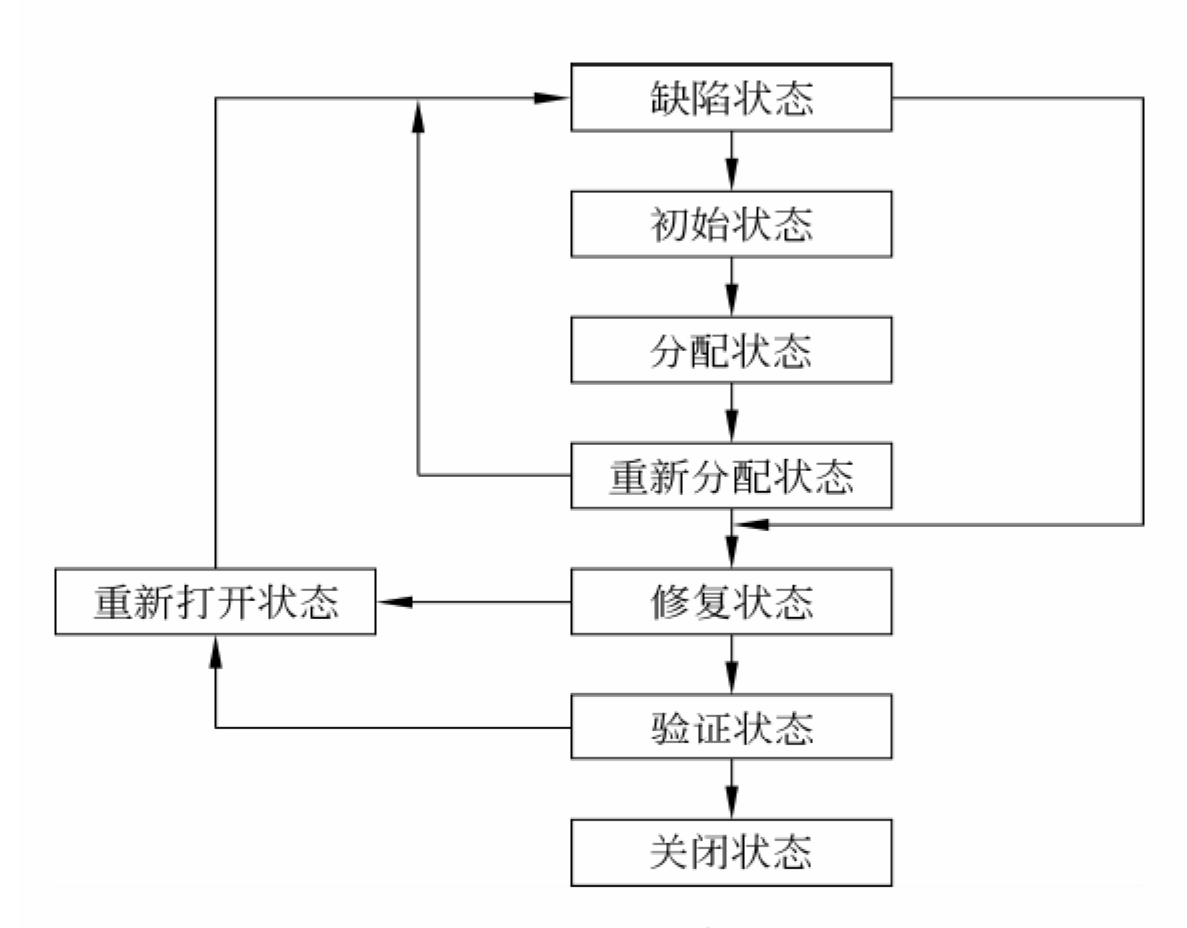


图 1.4 缺陷的生命周期图

1.3 习 题

1. 选择题

(1) 软件本身的特点和目前软件开发模式使隐藏在软件内部的质量缺陷不可能完全避免，在下列关于导致软件质量缺陷的原因的描述中，不正确的是________。

A. 软件需求模糊以及需求的变更，从根本上影响着软件产品的质量

B. 目前广为采用的手工开发方式难于避免出现差错

C. 程序员编码水平低下是导致软件缺陷的最主要原因

D. 软件测试技术具有缺陷

(2) 缺陷产生的原因是________。

A. 交流不充分及沟通不畅、软件需求的变更、软件开发工具的缺陷

B. 软件的复杂性、软件项目的时间压力

C. 程序开发人员的错误、软件项目文档的缺乏

D. 以上都是

2. 判断题

(1) 缺乏有力的方法学指导和有效的开发工具的支持，往往是产生软件危机的原因之一。 (　　)

(2) 目前的绝大多数软件都不适合于快速原型技术。 (　　)

(3) 在程序运行之前没法评估其质量。 (　　)

(4) 下列哪些活动是项目?

探索火星生命迹象。 (　　)

向部门经理进行月工作汇报。 (　　)

开发新版本的操作系统。 (　　)

每天的卫生保洁。 (　　)

组织超级女声决赛。 (　　)

一次集体婚礼。 (　　)

3. 简答题

(1) 什么是软件? 软件经历了哪几个发展阶段?

(2) 软件的缺陷是什么? 其分类有哪些?



软件测试基本知识

本章介绍软件测试的发展历程、软件测试目的和原则，软件测试的分类，并就软件测试模型和测试用例等理论知识给出了详细讲解。

2.1 测试发展历程

软件测试伴随着软件的产生而产生。早在20世纪50年代，英国著名的计算机科学家图灵就给出了软件测试的原始含义。他认为，测试是程序正确性证明的一种极端实验形式。早期软件开发过程中，软件规模小，复杂程度低，软件开发过程相当混乱无序，软件测试含义也比较窄，等同于“调试”，目的是纠正软件的故障，常常由软件开发人员自己进行，针对机器语言和汇编语言，设计特定的测试用例，运行被测试程序，将所得结果与预期结果进行比较，从而判断程序的正确性。对测试的投入极少，测试介入也晚，常常是等到形成代码，产品已经基本完成时才进行测试

直到1957年，软件测试首次作为发现软件缺陷的活动，与调试区分开来。1972年，北卡罗来纳大学举行首届软件测试会议，John Good Enough和Susan Gerhart在IEEE上发表《测试数据选择的原理》，确定软件测试是软件的一个研究方向。1975年，John Good Enough首次提出软件测试理论，从而把软件测试这一实践性很强的学科提高到理论的高度。1979年，Glenford Myers在《软件测试艺术》一书中，提出“测试是为发现错误而执行的一个程序或者系统的过程”。

20世纪80年代早期，软件和IT行业进入了大发展，软件趋向大型化、高复杂度，软件的质量越来越重要。一些软件测试的基础理论和实用技术开始形成，软件开发的方式也逐渐由混乱无序的开发过程过渡到结构化的开发过程，以结构化分析与设计、结构化评审、结构化程序设计以及结构化测试为特征，软件测试性质和内容也随之发生变化，测试不再是一个单纯的发现错误的过程，而是具有软件质量评价的内容。软件工程的概念逐步形成，软件开发模型产生。1983年，Bill Hetzel在《软件测试完全指南》中指出，测试是以评价一个程序或者系统属性为目标的任何一种活动，是对软件质量的度量。IEEE给软件测试定义为：“使用人工或自动手段来运行或测定某个软件系统的过程，其目的在于检验它是否满足规定的需求或弄清预期结果与实际结果直接的差别”。这个定义明确地指出，软件测试的目的是为了检验软件系统是否满足需求，软件测试不再是一个一次性的，也不只是开发后期的活动，而是与整个开发流程融合成一体。1986年L. J. Hayes提

出规格说明指导的模块测试方法，提出应在软件开发早期考虑测试的需求。

20 世纪 90 年代，随着面向对象分析和面向对象设计技术的日渐成熟，面向对象软件测试技术逐渐受到人们重视。1989 年 Fiedler 从面向对象的测试与传统测试的不同点出发，提出了面向对象单元测试的解决方案，从此开始了面向对象软件测试的研究工作。1994 年 9 月 Communication OF ACM 出版了面向对象的软件测试专集，涉及了类测试、集成测试和面向对象软件的课测试性等问题。

1996 年，测试能力成熟度 TMM、测试支持度 TSM、测试成熟度 TMM 等一系软件测试相关理论提出。到了 2002 年，Rick 和 Stefan 在《系统的软件测试》一书中对软件测试做了进一步描述：测试是为了度量和提高软件的质量，对软件进行工程设计、实施和维护的整个生命周期过程。近 20 年来，随着计算机和软件技术飞速发展，软件测试技术的研究也取得了很大的突破。许多测试模型（如 V 模型等）产生，单元测试、自动化测试等方面涌现了大量的软件测试工具。在软件测试工具平台方面，商业化的软件测试工具，如捕获/回放工具、Web 测试工具、性能测试工具、测试管理工具、代码测试工具等产生很多，一些开放源码社区中也出现了许多软件测试工具，得到了广泛应用且相当成熟和完善。

2.2 测试的几种观点

下面给出几种关于软件测试的观点。

1. 软件测试的广义论与狭义论

传统瀑布模型认为测试是指在代码完成后、处于运行维护阶段之前，通过运行程序来发现程序代码或软件系统中错误。因此，不能在代码完成之前发现软件系统需求以及设计上的缺陷，如果需求和设计上的缺陷问题遗留到后期，这样就会造成大量返工，增加软件开发的成本、延长开发的周期等。

为了更早地发现问题，将测试延伸到需求评审、设计审查活动中去，将“软件质量保证”的部分活动归为测试活动。软件生命周期每一阶段中都应包含测试，用于检验本阶段的成果是否接近预期的目标，尽可能早的发现错误并加以修正，将软件测试和质量保证合并起来的软件测试，被认为是一种软件测试的广义概念。

2. 软件测试的辩证论

验证软件是验证软件是“工作的”，是指软件的功能是按照预先的设计执行的，以正向思维针对软件系统的所有功能点，逐个验证其正确性。其代表人物是软件测试领域的先驱 Dr. Bill Hetzel（代表论著 *The Complete Guide to Software Testing*）。他曾于 1972 年 6 月在美国的北卡罗来纳大学组织了历史上第一次正式的关于软件测试的论坛。在 1973 年给软件测试一个这样的定义：“就是建立一种信心，认为程序能够按预期的设想运行。”其后，在 1983 年，他又将定义修改为：“评价一个程序和系统的特性或能力，并确定它是否达到预期的结果。软件测试就是以此为目的的任何行为。”

反之，证明软件是“不工作的”，以反向思维的方式，不断思考开发人员理解的误区、不

良的习惯、程序代码的边界、无效数据的输入以及系统的弱点，试图破坏系统、摧毁系统，目标就是发现系统中各种各样的问题。其代表人物是 Glenford J. Myers(代表论著 *The Art of Software Testing*)。他强调一个成功的测试必须是发现 Bug(缺陷)的测试，不然就没有价值。他于 1979 年提出了他对软件测试的定义："就是以发现错误为目的而运行程序的过程。"

3. 软件测试的风险论

软件测试的风险论认为测试是对软件系统中潜在的各种风险进行评估的活动。对应这种观点，产生基于风险的测试策略，首先评估测试的风险，功能出问题的概率有多大？哪些是用户最常用的 20%功能(Pareto 原则，也叫 80/20 原则)？如果某个功能出问题，其对用户的影响有多大？然后根据风险大小确定测试的优先级。优先级高的测试，优先得到执行。一般来讲，针对用户最常用的 20%功能(优先级高)的测试会得到完全执行，而低优先级的测试(另外用户不经常用的 80%功能)就不做或少做。

4. 软件测试的经济论

"一个好的测试用例是在于它能发现至今未发现的错误"，体现了软件测试的经济学观点。这是由于在需求阶段修正一个错误的代价是 1，而在设计阶段就是它的 3～6 倍，在编程阶段是它的 10 倍，在内部测试阶段是它的 20～40 倍，在外部测试阶段是它的 30～70 倍，而到了产品发布出去时，这个数字就是 40～1000 倍。修正错误的代价不是简单地随着时间线性增长，而几乎是呈指数级增长的。因此，应该尽快、尽早地发现缺陷。

5. 软件测试的标准论

软件测试的标准论认为软件测试为"验证(Verification)"和"有效性确认(Validation)"活动构成的整体，即软件测试等于 V&V。验证是检验软件是否已正确地实现了产品规格书所定义的系统功能和特性。有效性确认是确认所开发的软件是否满足用户真正需求的活动。

综上所述，软件测试是贯穿整个软件开发生命周期、是对软件产品进行验证和确认的活动过程，其目的是尽快尽早地发现在软件产品中所存在的各种问题。为了深入理解软件测试，可以从下面不同的角度来思考。

(1) 从软件测试的目的来理解。测试的目的是发现软件中的错误，是为了证明软件有错，而不是证明软件无错，是在软件投入运行前，对软件需求分析、设计和编码各阶段产品的最终检查，是为了保证软件开发产品的正确性、完全性和一致性。

(2) 从软件测试的性质来理解。在软件开发过程中，分析、设计与编码等工作都是"建设性的"，唯独测试是带有"破坏性的"。

(3) 从软件开发角度来理解。软件测试以检查软件产品的内容和功能特性为核心，是软件质量保证的关键步骤，也是成功实现软件开发目标的重要保障。

(4) 从软件工程角度来理解。软件测试是软件工程的一部分，是软件工程过程中的重要阶段。

(5) 从软件质量保证角度来理解。软件测试是软件质量保障的关键措施。

2.3 软件测试目的与原则

测试使用人工或者自动手段来运行或测试某个系统的过程，其目的在于检验它是否满足规定的需求或弄清预期结果与实际结果之间的差别。测试是帮助识别开发完成(中间或最终的版本)的计算机软件(整体或部分)的正确度、完全度和质量的软件过程，是保证软件质量的重要手段。

Grenford J. Myers 曾对软件测试的目的提出过以下观点：

(1) 测试是为了证明程序有错，而不是证明程序无错误。

(2) 一个好的测试用例是在于它能发现至今未发现的错误。

(3) 一个成功的测试是发现了至今未发现的错误的测试。

Grenford J. Myers 认为测试是以查找错误为中心，而不是为了演示软件的正确功能，从字面意思理解，可能会产生误导，认为发现错误是软件测试的唯一目的，查找不出错误的测试就是没有价值的测试。

软件测试的目的往往包含如下内容：

(1) 测试并不仅仅是为了找出错误，通过分析错误产生的原因和错误的发生趋势，可以帮助项目管理者发现当前软件开发过程中的缺陷，以便及时改进。

(2) 测试帮助测试人员设计出有针对性的测试方法，改善测试的效率和有效性。

(3) 没有发现错误的测试也是有价值的，完整的测试是评定软件质量的一种方法。

测试的目标就是以最少的时间和人力找出软件中潜在的各种错误和缺陷，证明软件的功能和性能与需求说明相符。此外，实施测试收集到的测试结果数据为可靠性分析提供了依据。

软件测试从不同的角度出发有两种测试原则。从用户的角度出发，就是希望通过软件测试能充分暴露软件中存在的问题和缺陷，从而考虑是否可以接受该产品；从开发者的角度出发，就是希望测试能表明软件产品不存在错误，已经正确地实现了用户的需求，确立人们对软件质量的信心。为了达到上述目的，需要注意以下几点原则：

1. 应当把“尽早地和不断地进行软件测试”作为软件开发者的座右铭

由于软件的复杂性和抽象性使得开发的每个环节都可能产生错误，所以不应把软件测试仅仅看作是软件开发的一个独立阶段，而应当把它贯穿到软件开发的各个阶段中，坚持在软件开发的各个阶段进行技术评审，尽早发现和预防错误，把出现的错误克服在早期，杜绝某些隐患，提高软件质量。

2. 严防寄生虫现象

软件问题会像寄生虫一样过群居生活，也称为群集现象。经验表明，测试后程序中残存的错误数目与该程序中已发现的错误数目成正比，这种错误群集性现象，已为许多程序的测试实践所证实。例如，IBM 公司开发 OS/370 操作系统时，大量的错误仅与该系统的

4%的模块有关,形成群集现象。因此,如果发现某一模块似乎比其他模块有更多的错误倾向时,应当对错误群集的程序段进行重点测试,以提高测试投资的效益。

3. 严防杀虫剂现象

农业生产中,长时期使用一种药物,害虫会产生抗药性,使得杀虫剂不起作用,这就是杀虫剂现象。在软件测试中则表现为当测试工作开展一段时间后,会找不到软件缺陷,这是因为在同一种思路下大部分软件缺陷问题已经被发现了,此时应改变测试方法和思路。换一种"杀虫剂",就会发现软件的缺陷就会再次出现。

4. 并非所有的软件缺陷都能修复

由于时间、人员、设备和资金等各种客观条件的限制,软件缺陷即使被发现后,也不可能都被修复,测试工作量与软件缺陷的数量关系如图 2.1 所示,因此,应当对故障集中的程序段进行重点测试。

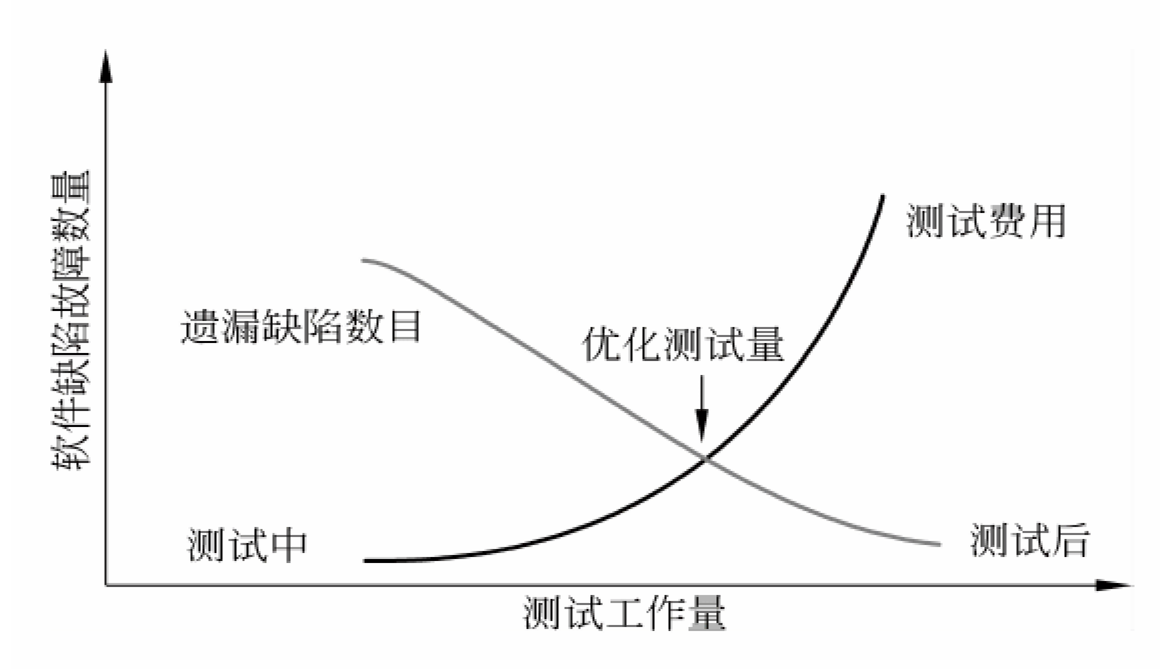


图 2.1 测试工作量与软件缺陷数量之间的关系

5. 难以说清的软件缺陷

软件测试是以需求规格说明书为标准,当需求规格说明书转化为测试需求后,才能作为测试工作判断缺陷的标准,这个转化过程会有人为理解因素,因此,由于需求不明确会导致缺陷的产生,使得软件测试具有复杂性。

6. 测试用例的设计

测试用例应由测试输入数据和与之对应的预期输出结果这两部分组成。测试用例的设计应考虑到合法的输入和非法输入以及各种边界条件、特殊情况等,合理的输入条件是指能验证程序正确的输入条件,而不合理的输入条件是指异常的、临界的、可能引起问题异变的输入条件。用不合理的输入条件测试程序时,往往比用合理的输入条件进行测试能发现更多的错误。

7. 软件测试充分性准则

(1) 对任何软件都存在有限的充分测试集合。

(2) 如果一个软件系统在一个测试数据集合上的测试是充分的，那么再多测试一些数据也应该是充分的。这一特性称为单调性。

(3) 即使对软件所有成分都进行了充分的测试，也并不表明整个软件的测试已经充分了。这一特性称为非复合性。

(4) 即使对软件系统整体的测试是充分的，也并不意味软件系统中各个成分都已经充分地得到了测试。这个特性称为非分解性。

(5) 软件测试的充分性应该与软件的需求和软件的实现都相关。

(6) 软件越复杂，需要的测试数据就越多。这一特性称为复杂性。

(7) 测试得越多，进一步测试能得到的充分性增长越少。这一特性称为回报递减率。

2.4 软件测试分类

软件测试的分类如图 2.2 所示。

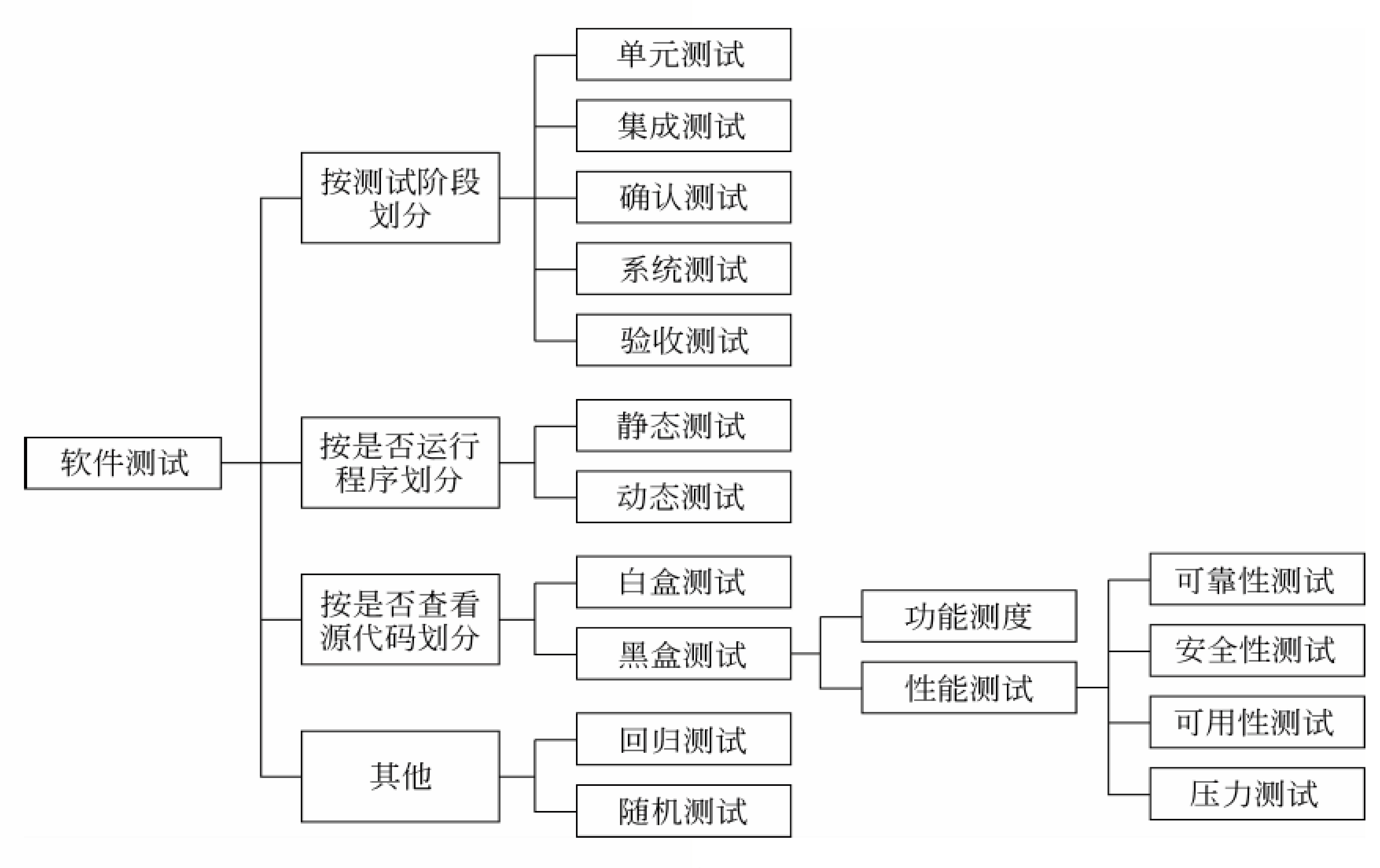


图 2.2 软件测试分类

2.4.1 按照测试阶段划分

软件测试贯穿整个软件开发的整个期间，按照软件测试阶段划分，软件测试分为单元测试、集成测试、确认测试、系统测试、验收测试等。

- 单元测试用于检验被测代码的一个很小的、很明确的功能是否正确。通常，单元测试用于判断某个特定条件下某个特定函数的行为。
- 集成测试是指将经过单元测试的模块之间的依赖接口的关系图进行的测试。
- 确认测试用于验证软件的有效性，即验证软件的功能和性能及其他特性是否与用户的要求一致。

- 系统测试将作为整个软件系统与计算机硬件、外设、支持软件、数据和人员等其他系统元素结合起来进行测试。
- 验收测试是指最终用户参与测试的过程。

2.4.2 按照执行主体划分

按照测试实施组织划分，软件测试分为 α 测试、β 测试和第三方测试。

1. α 测试

通常也叫“验收测试”或“开发方测试”，在软件开发环境中，开发者和用户共同去检测与证实软件的实现是否满足软件设计说明或软件需求说明的要求。

2. β 测试

通常 β 测试被认为是用户测试，是指用户的使用性测试，由用户找出软件在应用过程中发现的软件的缺陷与问题，并对使用质量进行评价。

3. 第三方测试

第三方测试也称独立测试，是由第三方测试机构来进行的测试。由与开发方和用户方都相对独立的组织进行软件测试，通过模拟用户真实的环境，进行确认测试。

2.4.3 按照执行状态划分

按照测试执行状态划分，软件测试分为动态测试和静态测试。

1. 动态测试

软件的动态测试，是指通过运行被测程序，检查运行结果与预期结果的差异，并分析运行效率和健壮性等性能，这种方法由三部分组成：构造测试实例、执行程序、分析程序的输出结果。

2. 静态测试

静态测试是指不真正运行被测试的程序，通过人工对程序和文档进行分析与检查，包括走查、符号执行、需求确认等。静态测试一方面利用计算机作为对被测程序进行特性分析的工具，与人工测试有着根本的区别；另一方面并不真正运行被测程序，与动态方法也不相同。因此，静态测试常称为“分析”，是对被测程序进行特性分析方法的总称。

静态分析过程如图 2.3 所示。针对代码的静态测试包括代码检查、静态结构分析、代码质量度量等。具体如下所示。

1）代码检查

代码检查主要检查代码和设计的一致性，代码对文档标准的遵循及代码的可读性，代码的逻辑表达正确性，代码结构的合理性等方面。代码检查比动态测试更有效率，能快速找到大约 30％～70％的逻辑设计错误和编码缺陷。代码检查一般在编译和动态测试之

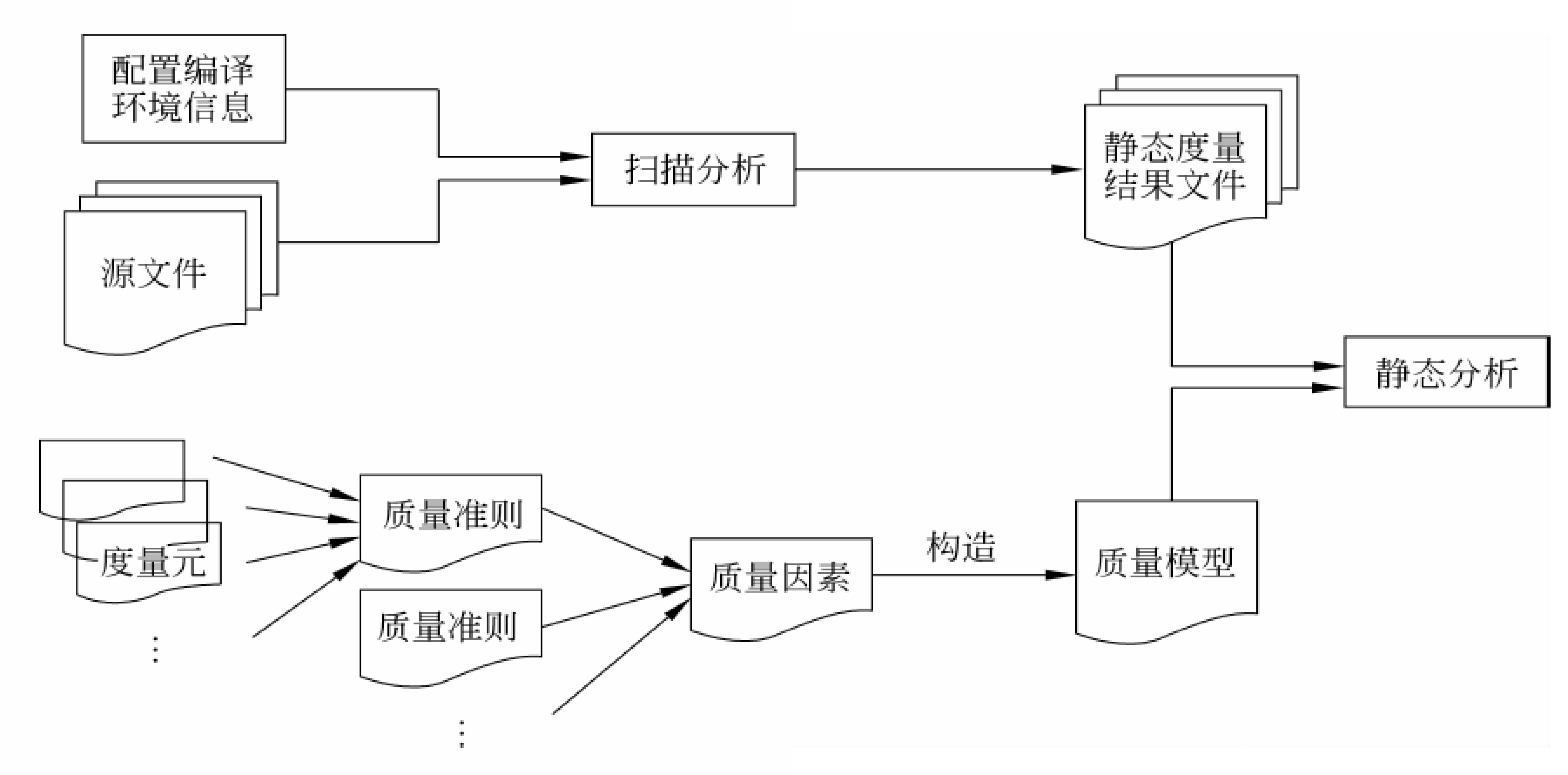


图 2.3　静态分析过程

前进行，其实施方法很多，如走查、审查或评审等，如表 2.1 所示。

表 2.1　走查、评审、伙伴检查的对比

事项	走　　查	审查(评审)	伙伴检查
准备	通读设计和编码	应准备好需求描述文档、程序设计文档、程序的源代码清单、代码编码标准和代码缺陷检查表	没有准备
形式	非正式会议	正式会议	
主持人	任何人	由非该软件的编制人员组成	没有
参加人员	2～7 人，以开发人员为主	3～6 人，项目组成员及测试人员	1 或 2 人
主要技术方法	无	缺陷检查表	无
注意事项	限时、不要现场修改代码	限时、不要现场修改代码	无
生成文档	会议记录	静态分析错误报告	口头评论
目标	代码标准规范、无逻辑错误	代码标准规范、无逻辑错误	无
优点	能使更多人熟悉产品	费用低	费用低
缺点	查出的故障较少	短期成本高	查出的故障较少

(1) 走查

走查是在开发组内部进行，不像软件审查那么正式，准备工作一般由主持人负责，参加人员只需要简单地参加会议，通过个人检查和阅读等手段来查找错误。其主要检查逻辑错误和代码是否符合标准、规范和风格等错误，具有不在现场修改等特点。

(2) 审查

审查，又称评审，由开发组、测试组和相关人员(QA、产品经理等)联合进行，通过分配相关的角色，采用讲解、提问并使用检查表方式进行的查找错误活动。审查通过会议的

形式,一般参加人员包括与本模块相关的开发人员,由一名开发者讲解、其他开发者提问,本模块开发者回答问题,填写检查表。一般有正式的计划、流程和结果报告。其中,同行(对等)评审是指由与工作产品开发人员具有同等背景和能力的人员对工作产品进行的评审,其目的是有效地消除软件的缺陷。

(3) 伙伴检查

由于软件审查短期费用往往比较高,因此,常常只挑选几个人,针对核心代码等采用伙伴检查的形式进行,通常是在编写代码的程序员和充当审查者的其他一两个程序员或测试人员之间进行。

2) 静态结构分析

静态结构分析以图形的方式表现程序的内部结构,例如,函数调用关系图、函数内部控制流图等。其中,函数调用关系图描述程序中函数调用与被调用的关系,控制流图显示函数的逻辑结构。

3) 代码质量度量

代码质量主要有 3 种度量方式:Line 复杂度、Halstead 复杂度和 McCabe 复杂度。其中,Line 复杂度以代码的行数作为计算的基准。Halstead 以程序中使用到的运算符与运算元的数量作为计数目标,然后计算出程序的工作量。McCabe 复杂度又称圈复杂度,是将程序的流程图转化为有向图,用图论来计算软件的复杂度。

2.4.4 按照测试技术划分

按照对于被测的对象了解划分,软件测试分为黑盒测试、白盒测试和灰盒测试。

1. 黑盒测试

黑盒测试也称功能测试或数据驱动测试,它是在已知产品所应具有的功能,通过测试来检测每个功能是否都能正常使用,在测试时,把程序看做一个不能打开的黑盆子,在完全不考虑程序内部结构和内部特性的情况下,测试者在程序接口进行测试,只检查程序功能是否按照需求规格说明书的规定正常使用,程序是否能适当地接收输入数据而产生正确的输出信息,并且保持外部信息(如数据库或文件)的完整性。

黑盒测试试图发现以下类型的错误:功能错误或遗漏、界面错误、数据结构或外部数据库访问错误、性能错误、初始化和终止错误等。

2. 白盒测试

白盒测试与黑盒测试正好相反,又称结构测试或逻辑驱动测试,它是知道产品内部工作过程,检测产品内部动作是否按照规格说明书的规定正常进行,按照程序内部的结构测试程序,检验程序中的每条通路是否都有能按预定要求正确工作,白盒测试的主要方法有逻辑驱动、路径测试等,主要用于软件验证。白盒测试是基于源代码下的测试,需要了解程序的构架、具体需求以及一些编写程序的技巧,能够检查一些程序规范、指针、变量、数

组越界等问题。

白盒测试容易发现以下类型的错误：变量没有声明、无效引用、数组越界、死循环、函数本身没有析构、参数类型不匹配、调用系统的函数没有考虑到系统的兼容性等。

3. 灰盒测试

灰盒测试介于黑盒测试和白盒测试之间，主要用于测试各个组件之间的逻辑关系是否正确，采用桩驱动，把各个函数按照一定的逻辑串起来，达到在产品还没有界面的情况下的结果输出。灰盒测试相对白盒测试来说要求相对较低，对测试用例要求也相对较低，用于代码的逻辑测试，验证程序接收和处理参数。灰盒测试的重点在于测试程序的处理能力和健壮性，相对黑盒测试和白盒测试而言，投入的时间相对少，维护量也较小。

软件测试方法与软件开发过程相关联，单元测试一般采用白盒测试方法，集成测试采用灰盒测试方法，系统测试和确认测试采用黑盒测试方法。

黑盒测试和白盒测试的比较如表 2.2 所示。

表 2.2　黑盒测试和白盒测试比较

项目	黑盒测试法	白盒测试法
规划方面	功能测试	结构测试
性质	是一种确认（Validation）技术，回答“我们在构造一个正确的系统吗?”	是一种验证（Verification）技术，回答“我们在正确地构造一个系统吗?”
优点方面	（1）确保从用户角度出发 （2）适用于各阶段测试 （3）从产品功能角度测试 （4）容易入手生成测试数据	（1）针对程序内部特定部分进行覆盖测试 （2）可构成测试数据使特定程序部分得到测试 （3）有一定的充分性度量手段 （4）可或较多工具支持
缺点方面	（1）无法测试程序内部特定部分 （2）某些代码得不到测试 （3）如果规格说明有误，则无法发现 （4）不易进行充分性测试	（1）无法测试程序外部特性 （2）不易生成测试数据（通常） （3）无法对未实现规格说明的部分进行测试 （4）工作量大，通常只用于单元测试
应用范围	边界分析法、等价类划分法、决策表测试	语句覆盖、判定覆盖、条件覆盖、路径覆盖等

如图 2.4 所示，软件测试的三维空间是一个相当复杂的过程，包括测试目标、测试方法和测试阶段。下面依次进行介绍。

1）测试目标

测试目标用于验证软件质量特性，包括功能测试、强壮性测试、性能测试、适用性测试、安全性测试和可靠性测试等。

2）测试方法

测试方法分为白盒测试、灰盒测试和黑盒测试。这与人类分析问题和解决问题有关，当人类对被测的对象/世界（软件）认知很少，不了解其内部结构，只关注其外部的变化，如外部的输入、外部作用或被测的对象所处的条件以及被测的对象输出的结果，就是黑盒测试方法。随着对被测的对象的认知越来越多，就可以采用灰盒测试方法。当人类能完全

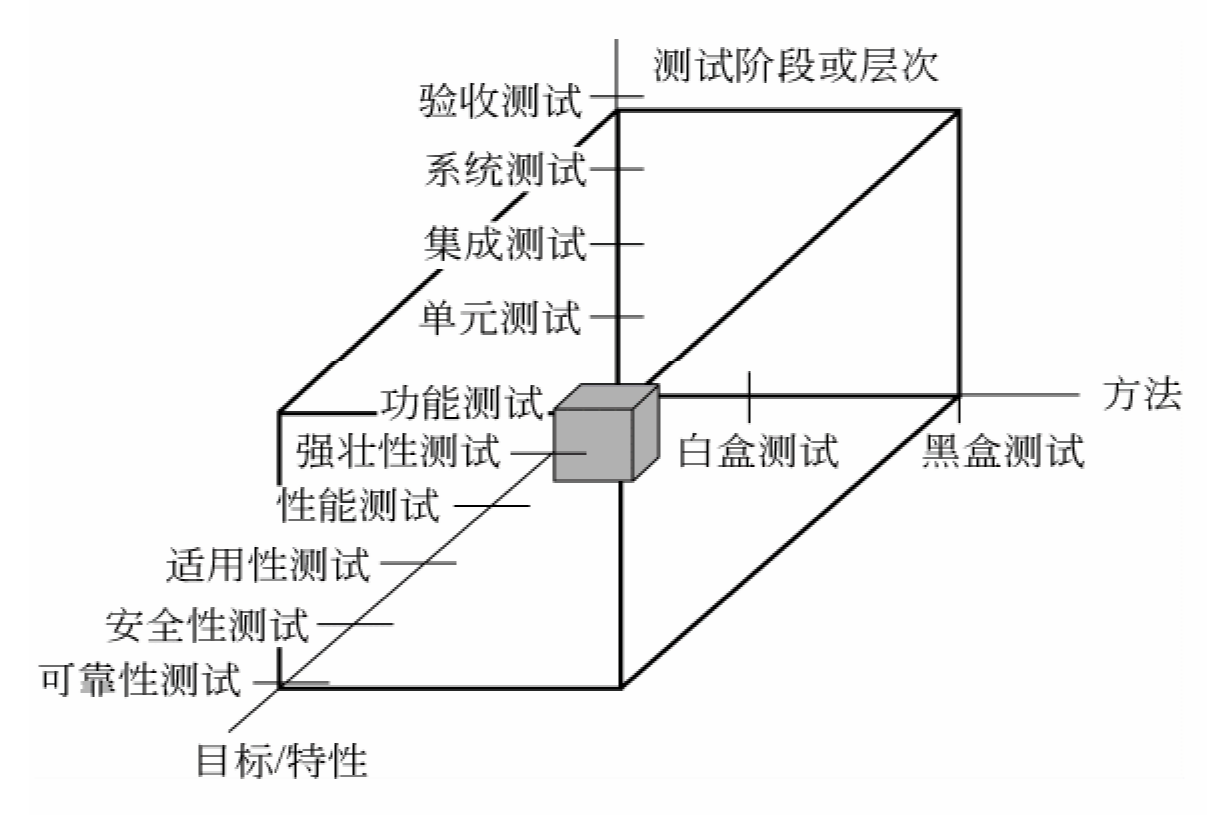


图 2.4　软件测试的三维空间

认知被测的对象的内部结构是，就是白盒测试方法。

3）测试阶段

软件开发的生命周期包含了各类活动，测试与之对应，也划分了不同的测试阶段，测试的步骤分为：需求分析审查、设计审查、单元测试、集成测试、系统测试、验收测试、回归测试等，各阶段的输入和输出如表 2.3 所示。

表 2.3　测试阶段的输入与输出

阶　段	输入和要求	输　　出
需求分析审查	市场/产品需求定义、分析文档和相关技术文档 要求：需求定义要准确，完整和一致，真正理解客户的需求	需求分析中问题列表，批准的需求分析文档、测试计划书的起草
设计审查	产品规格设计说明、系统架构和技术设计文档、测试计划和测试用例 要求：系统结构的合理性、处理过程的正确性、数据库的规范化、模块的独立性等 清楚定义测试计划的策略、范围、资源和风险，测试用例的有效性和完备性	设计问题列表、批准的各类设计文档，系统和功能的测试计划和测试用例 测试环境的准备
单元测试	源程序、编程规范、产品规格设计说明书和详细的程序设计文档 要求：遵守规范、模块的高内聚性、功能实现的一致性和正确性	缺陷报告、跟踪报告；完善的测试用例、测试计划，了解系统功能及其实现等
集成测试	通过单元测试的模块或组件、编程规范、集成测试规格说明和程序设计文档，系统设计文档 要求：接口定义清楚且正确，模块或组件一起工作正常，集成为完整的系统	缺陷报告、跟踪报告；完善的测试用例、集成测试分析报告、集成后的系统
系统测试	软件包、策划四环境、系统测试用例和测试计划 要求：系统能正常地、有效的运行，包括性能、可靠性、安全性和兼容性等	缺陷报告、跟踪报告；完善的测试用例、阶段性测试报告

2.5 软件测试模型

软件测试模型用于指导软件测试的实践，通常有一些测试模型，如 V 模型、W 模型、H 模型、X 模型和前置模型等。下面依次进行介绍。

2.5.1 V 模型

V 模型作为最典型的测试模型，由 Paul Rook 在 20 世纪 80 年代后期提出，如图 2.5 所示。V 模型反映了测试活动与开发活动的关系，标明测试过程中存在的不同级别，并清楚描述测试的各个阶段和开发过程的各个阶段之间的对应关系。V 模型左侧是开发阶段，右侧是测试阶段。开发阶段先从定义软件需求开始，然后把需求转换为概要设计和详细设计，最后形成程序代码。测试阶段是在代码编写完成以后，先作单元测试开始，然后是集成测试、系统测试和验收测试。在 V 模型中，单元测试对应详细设计，也就是说，单元测试用例和详细设计文档一起实现；而集成测试对应于概要设计，其测试用例是根据概要设计中模块功能及接口等实现方法编写。依次类推，测试计划在软件需求完成后就开始进行，完成系统测试用例的设计等。

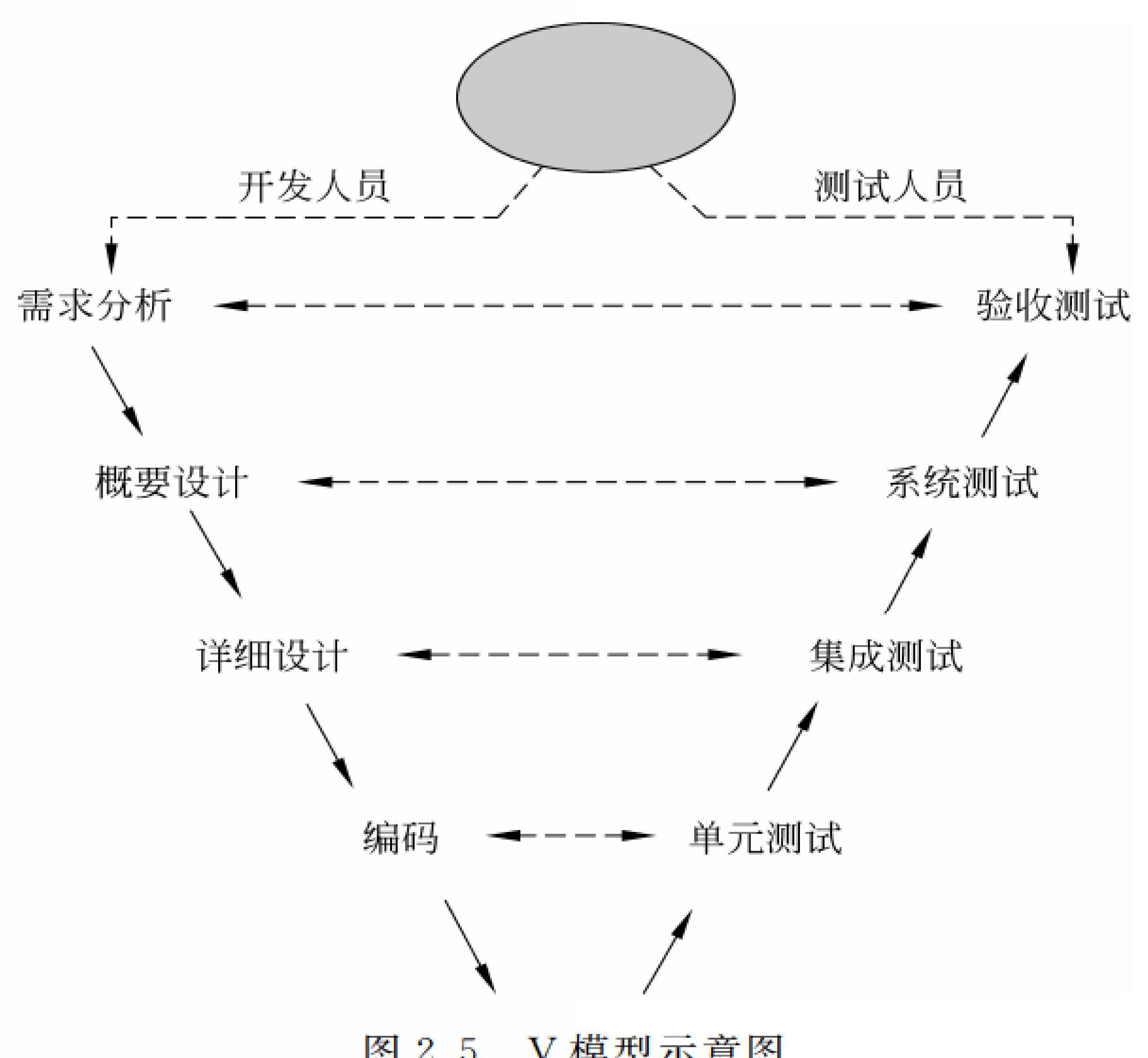


图 2.5　V 模型示意图

V 模型仅把测试过程作为在需求分析、概要设计、详细设计及编码之后的一个阶段，主要针对程序进行寻找错误的活动，而忽视了测试活动对需求分析、系统设计等活动的验证和确认的功能。

2.5.2 W 模型

相对于 V 模型而言，W 模型增加了软件各开发阶段中应同步进行的验证和确认活

动。如图 2.6 所示,W 模型由两个 V 字型模型组成,分别代表测试与开发过程,明确地表示出了测试与开发的并行关系。

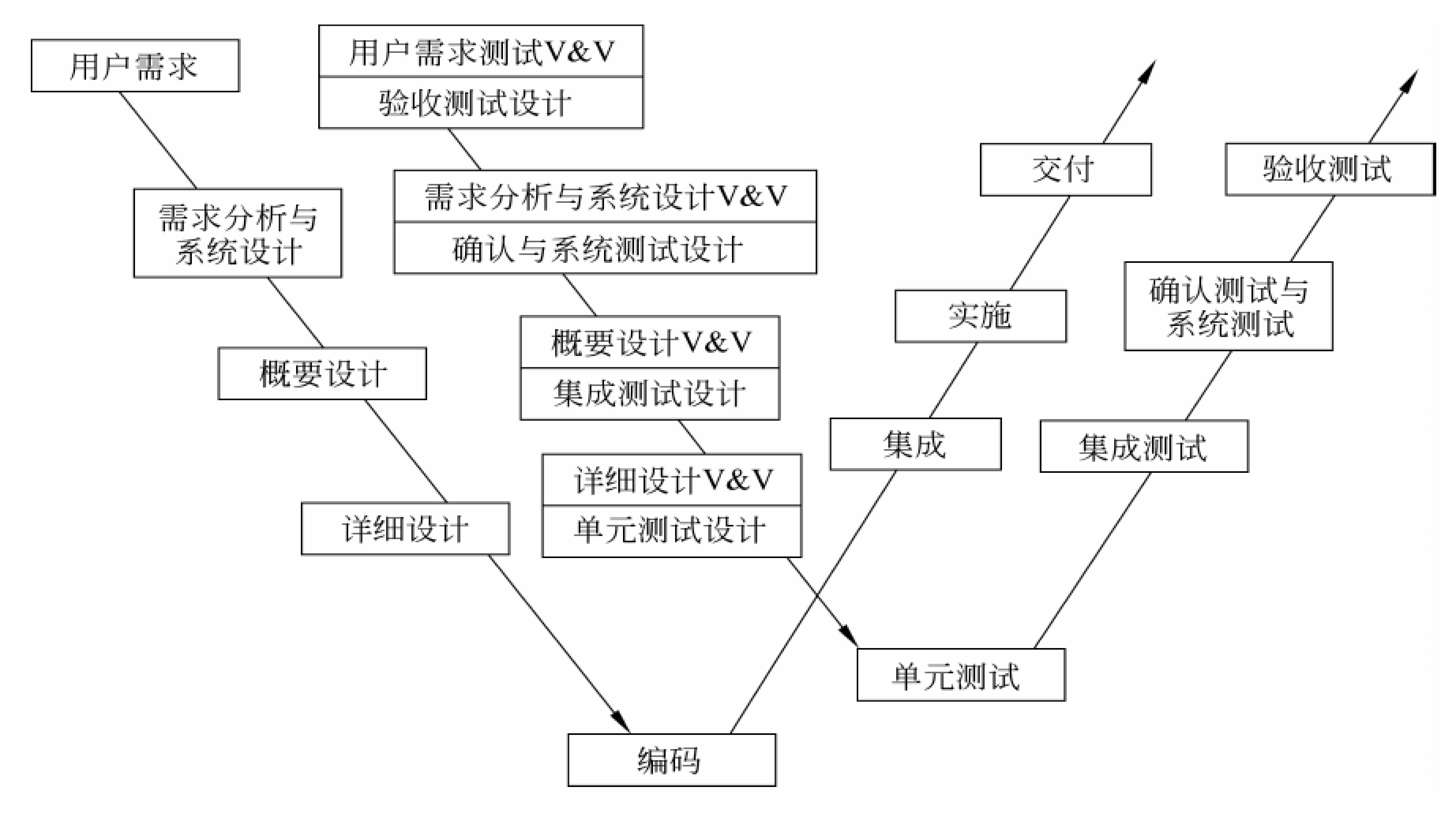


图 2.6　W 模型示意图

W 模型强调,测试伴随着整个软件开发周期,测试的对象不仅仅是程序,需求、设计等同样要测试,也就是说,测试与开发同步进行。W 模型有利于尽早地发现问题,只要相应的开发活动完成,就可以开始测试。例如,需求分析完成后,测试就应该参与到对需求的验证和确认活动中,以尽早找出缺陷所在。同时,对需求的测试也有利于及时了解项目难度和测试风险,及早制定应对措施,从而减少总体测试时间,加快项目进度。

在 W 模型中,需求、设计、编码等活动被视为串行,测试和开发活动保持着一种线性的前后关系,上一阶段结束,才开始下一个阶段工作,这是 W 模型存在的局限性,因此,W 模型无法支持迭代开发模型。

2.5.3　H 模型

V 模型和 W 模型都认为软件开发是需求、设计、编码等一系列串行的活动,而事实上,这些活动在大部分时间内可以交叉,因此,相应的测试也不存在严格的次序关系,单元测试、集成测试、系统测试之间具有反复迭代。正因为 V 模型和 W 模型存在这样的问题,H 模型将测试活动完全独立出来,使得测试准备活动和测试执行活动清晰地体现出来。

图 2.7 仅仅显示了整个测试生命周期中某个层次的“微循环”。H 模型揭示了软件测试作为一个独立的流程贯穿于软件整个生命周期,与其他流程并发地进行,并指出软件测试要尽早准备,尽早执行。不同的测试活动可以按照某个次序先后进行,也可能是反复的,只要某个测试达到准备就绪点,测试执行活动就可以开展。

因此,H 模型具有如下意义:

(1) 测试准备与测试执行分离,有利于资源调配,降低成本,提高效率。

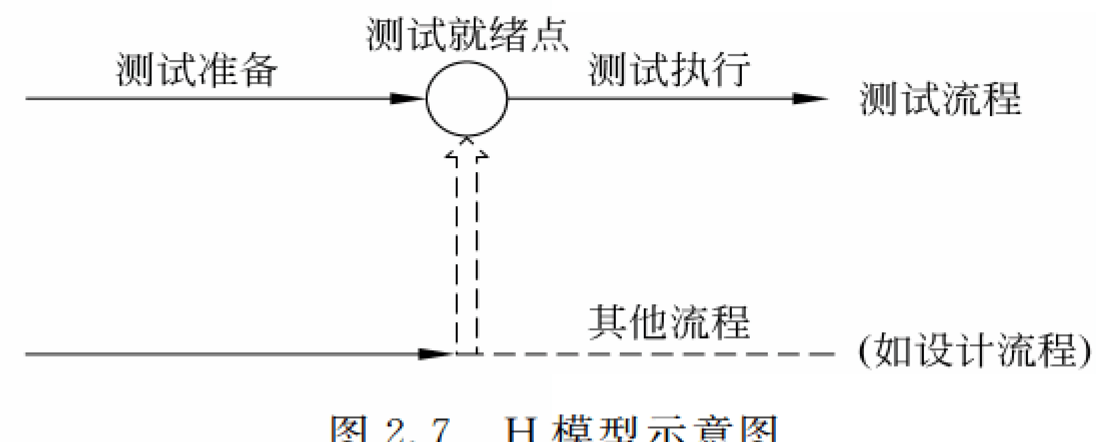


图 2.7　H 模型示意图

(2) 充分体现测试过程(不是技术)的复杂性。

2.5.4　X 模型

由于 V 模型没能体现出测试设计、测试回溯的过程,因此,出现了 X 测试模型,如图 2.8 所示。

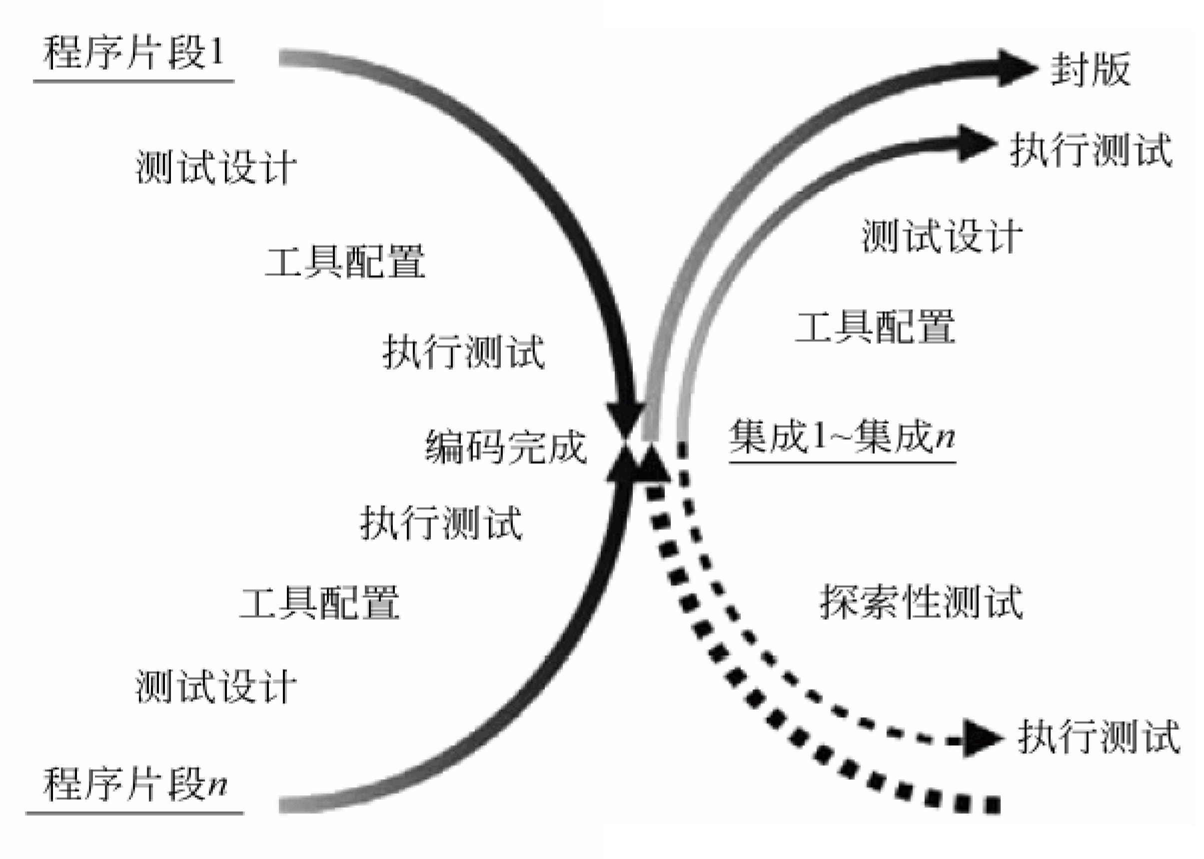


图 2.8　X 模型示意图

X 模型的左边描述的是针对单独程序片段所进行的编码和测试,此后将进行频繁的交接,通过集成最终合成为可执行的程序。X 模型右上方定位了已通过集成测试的成品进行封版并提交给用户,也可以作为更大规模和范围内集成的一部分。多根并行的曲线表示变更可以在各个部分发生。X 模型右下方定位了探索性测试。这是不进行事先计划的特殊类型的测试,往往帮助有经验的测试人员在测试计划之外发现软件错误。

2.5.5　前置模型

前置模型将测试和开发紧密结合,具有如下的优点。

1. 开发和测试相结合

前置测试模型将开发和测试的生命周期整合在一起,标识了项目生命周期从开始到结束之间的关键行为,表示这些行为在项目周期中的价值。前置测试在开发阶段以编码、

测试、编码、测试的方式进行。也就是说，程序片段编写完成，会进行测试。

2. 对每一个交付内容进行测试

每一个交付的开发结果都必须通过一定的方式进行测试。源程序代码并不是唯一需要测试的内容。可行性报告、业务需求说明，以及系统设计文档等也是被测试的对象。这同 V 模型中开发和测试的对应关系相一致，并且在其基础上有所扩展。

3. 让验收测试和技术测试保持相互独立

验收测试应该独立于技术测试，这样可以提供双重的保险，以保证设计及程序编码能够符合最终用户的需求。验收测试既可以在实施阶段的第一步来执行，也可以在开发阶段的最后一步执行。

4. 反复交替的开发和测试

项目开发中存在很多变更，例如需要重新访问前一阶段的内容，或者跟踪并纠正以前提交的内容，修复错误，增加新发现的功能等，开发和测试需要一起反复交替地执行。

5. 引入新的测试理念

前置测试对软件测试进行优先级划分，用较低的成本及早发现错误，并且充分强调了测试对确保系统高质量的重要意义。

总之，V 模型、W 模型、H 模型、X 模型以及前置模型都有各自的优点和缺点，应根据实际需要，灵活运用各种模型。表 2.4 给出了各种测试模型的特点。

表 2.4　测试模型的各自特点

模型	优　缺　点
V 模型	强调了整个软件项目开发中需要经历的若干个测试级别，每个级别都与一个开发阶段相对应，但它没有明确指出应该对需求、设计进行测试
W 模型	对 V 模型进行了补充。强调了测试计划等工作的先行和对系统需求和系统设计的测试，但和 V 模型一样，没有专门针对软件测试的流程予以说明
H 模型	表现了测试是独立的。就每一个软件的测试细节来说，都有一个独立的操作流程，只要测试前提具备了，就可以开始进行测试

2.6 测试用例

测试用例是指对一项特定的软件产品进行测试任务的描述，体现为测试方案、方法、技术和策略等，测试用例的内容包括测试目标、测试环境、输入数据、测试步骤、预期结果、测试脚本等，并形成文档。测试用例作为测试工作的指导，是软件测试的必须遵守的准则，是软件测试质量稳定的根本保障，其目的是能够将软件测试的行为转化成可管理的模

式，同时测试用例也是将测试具体量化的方法之一。

测试用例的重要性体现在如下几个方面。

(1) 测试用例构成了设计和制定测试过程的基础。

(2) 测试的"深度"与测试用例的数量成比例。由于每个测试用例反映不同的场景、条件或经由产品的事件流。

(3) 测试工作量与测试用例的数量成比例。根据全面且细化的测试用例，可以更准确地估计测试周期各连续阶段的时间安排。

(4) 测试设计和开发的类型以及所需的资源主要都受控于测试用例。

(5) 测试用例通常根据它们所关联关系的测试类型或测试需求来分类，而且将随类型和需求进行相应地改变。

测试用例主要有如下几种。

- 功能测试用例：包含功能测试。
- 性能测试用例：包含性能测试、压力测试、强度测试。
- 集成测试用例：包含接口测试、健壮性测试、可靠性测试。
- 安全测试用例。
- 用户界面测试用例。用户界面测试、少量功能测试。
- 安装/反安装测试用例。

软件测试种类、阶段和用例的关系如表 2.5 所示。

表 2.5　测试阶段与测试用例关系列表

测试阶段	测 试 类 型	执 行 人 员
单元测试	模块功能测试，包含部分接口测试、路径测试	开发人员
集成测试	接口测试、路径测试，含部分功能测试	开发人员，如果测试人员水平较高可以由测试人员执行
系统测试	功能测试、健壮性测试、性能测试、用户界面测试、安全性测试、压力测试、可靠性测试、安装/反安装测试	测试人员
验收测试	对于实际项目基本同上，并包含文档测试；对于软件产品主要测试相关技术文档	测试人员，可能包含用户

软件产品或软件开发项目的测试用例一般以该产品的软件模块或子系统为单位，形成一个测试用例文档。编写测试用例文档应有文档模板，须符合内部的规范要求。测试用例文档将受制于测试用例管理软件的约束。测试用例文档模板如表 2.6 所示。

表 2.6　测试用例文档模板

编制人		审定人		时间	
软件名称			编号/版本		
测试用例					
用例编号					

续表

参考信息(参考的文档及章节号或功能项):
输入说明(列出选用的输入项,列出预期输出):
输出说明(逐条与输入项对应,列出预期输出):
环境要求(测试要求的软、硬件、网络要求):
特殊规程要求:
用例间的依赖关系:
用例产生的测试程序限制:

测试用例文档由简介和测试用例两部分组成。简介部分编制了测试目的、测试范围、定义术语、参考文档、概述等。测试用例部分逐一列示各测试用例。每个具体测试用例都将包括下列详细信息:用例编号、用例名称、测试等级、入口准则、验证步骤、期望结果(含判断标准)、出口准则、注释等。以上内容涵盖了测试用例的基本元素:测试索引、测试环境、测试输入、测试操作、预期结果、评价标准。

测试用例可以分为基本事件、备选事件和异常事件的测试用例。设计基本事件的用例,应该参照用例规约,根据关联的功能、操作按路径分析法设计测试用例。而对孤立的功能则直接按功能设计测试用例。设计备选事件和异常事件的用例,则要复杂和困难得多。测试用例一般具有以下属性:

(1) 测试用例具有优先级。优先级越高,被执行的时间越早、执行的频率越多。由最高优先级的测试用例组来构成基本验证测试,每次构建软件包时,都要被执行一遍。

(2) 测试用例具有目标性。有的测试用例是为主要功能而设计,有的测试用例是为次要功能而设计,有的则为系统的负载而设计,有的则为一些特殊场合而设计。因此,需要根据不同的目标设计不同的测试用例。

(3) 测试用例具有范围性。测试用例属于哪一个组件或模块所有。

(4) 测试用例具有关联性。测试用例一般和软件产品特性相联系的,多数情况下验证某个产品的功能。

(5) 测试用例具有阶段性。测试用例应针对单元测试、集成测试、系统测试、验收测试阶段进行设计。这样对每个阶段,构造一个测试用例的集合被执行,并容易计算出该阶段的测试覆盖率。

(6) 测试用例具有状态性。当前是否有效,如果无效,被置于未被激活状态,不会被运行,只有被激活的测试用例才被运行。

(7) 测试用例具有代表性。测试用例能够代表并覆盖各种合理和不合理,合法和非法的,边界和越界的以及极限输入数据操作和环境设置等。

(8) 测试用例具有时效性。针对同样功能,可能所用的测试用例不同,是因为不同的产品版本在产品功能、特性等方面的要求不同。

根据上述特性,进行测试用例的编号、标题、描述(条件、步骤、期望结果)等设计,就可以对测试用例进行基于数据库方式的管理。

2.7 习　　题

1. 选择题

（1）软件测试按照测试技术划分为________。

A. 性能测试、负载测试、压力测试　　B. 恢复测试、安全测试、兼容测试

C. A 与 B 都是　　D. 单元测试、集成测试、验收测试

（2）软件测试的目的是________。

A. 避免软件开发中出现的错误

B. 发现软件开发中出现的错误

C. 尽可能发现并排除软件中潜藏的错误，提高软件的可靠性

D. 修改软件中出现的错误

（3）各个地方对软件测试定义不同，请根据软件测试方面、理论方面、代码的角度测试填空。

代码方面分为：________、集成测试、系统测试、验收测试（Alpha、Beta）

理论方面分为：________、动态测试、静态测试

测试方面分为：________、压力测试、回归测试、恢复测试、安全性测试、兼容性测试、内存泄露测试、比较测试等。

A. 单元测试　　B. 黑盒测试　　C. 白盒测试　　D. 负载测试

2. 判断题

（1）Beta 测试是验收测试的一种。　（　　）

（2）尽量用公共过程或子程序去代替重复的代码段。　（　　）

（3）测试是为了验证该软件已正确地实现了用户的要求。　（　　）

（4）发现错误多的程序模块，残留在模块中的错误也多。　（　　）

（5）尽量采用复合的条件测试，以避免嵌套的分支结构。　（　　）

3. 简答题

（1）软件测试的目的是什么？

（2）软件测试中应注意哪些事项？

（3）按执行主体划分，软件测试分哪几类？

（4）V 模型和 W 模型各自的优缺点是什么？

（5）测试用例是什么？有什么属性？



软件测试流程

本章详细地介绍了软件测试的整个过程，包括测试计划、测试设计、测试执行以及测试评估。其中，测试执行又包括单元测试、集成测试、确认测试、系统测试和验收测试等。

3.1 测试流程概述

软件测试流程与软件开发流程类似，也包括测试计划、测试设计、测试开发、测试执行和测试评估几个部分，如图 3.1 所示。

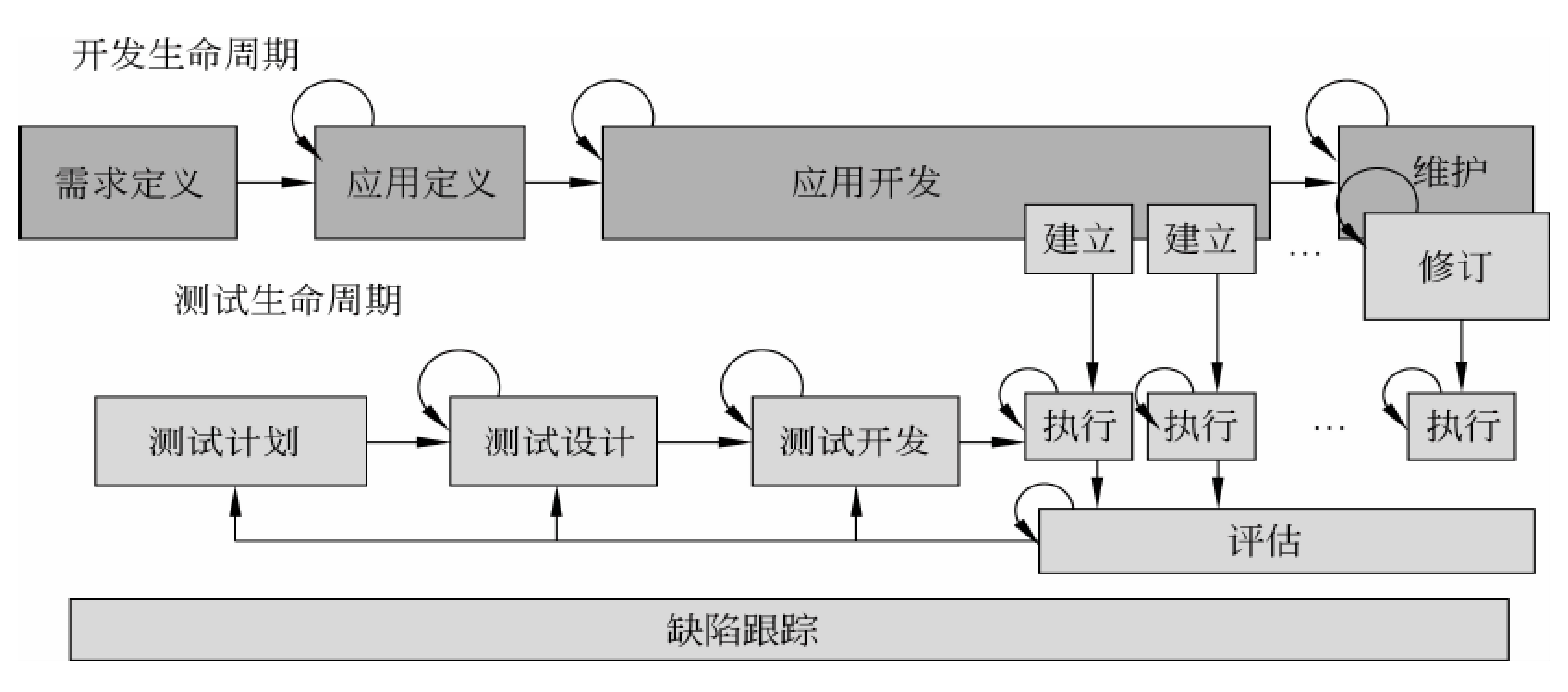


图 3.1　测试生命周期

软件测试生命周期具体如下所示。

1. 测试计划

根据用户需求报告中关于功能要求和性能指标的规格说明书，定义相应的测试需求报告，使随后所有的测试工作都围绕着测试需求来进行。同时，适当选择测试内容，合理安排测试人员、测试时间及测试资源等。

2. 测试设计

测试设计是指将测试计划阶段制订的测试需求分解、细化为若干个可执行的测试过程，并为每个测试过程选择适当的测试用例，保证测试结果的有效性。

3. 测试执行

测试执行开发阶段建立的自动测试过程，并对所发现的缺陷进行跟踪管理。测试执行一般由单元测试、组合测试、集成测试以及回归测试等步骤组成。

4. 测试评估

测试评估产生了具有测试覆盖域及缺陷跟踪报告，对于应用软件的质量和开发团队的工作进度及工作效率进行综合分析。

3.2 测试计划

测试计划由测试负责人来编写，用于确定各个测试阶段的目标和策略。这个过程将输出测试计划，明确要完成的测试活动，评估完成活动所需要的时间和资源，进行活动的安排和资源分配等。

测试计划的依据主要是项目开发计划和测试需求分析结果而制定。测试计划一般包括测试背景、测试依据、测试资源、测试策略和测试日程等内容，具体内容如下所示。

1. 测试背景

测试背景包括软件项目介绍；项目涉及人员（如软硬件项目负责人等）介绍以及相应联系方式等。

2. 测试依据

测试依据包括软件需求文档、软件规格书、软件设计文档等。

3. 测试资源

测试资源包括测试设备需求、测试人员需求、测试环境需求等。

4. 测试策略

测试策略包括采取测试方法、搭建测试环境、采取测试工具以及测试管理工具等的选择，还包括对测试人员进行培训等。

5. 测试日程

测试日程包括测试需求分析、测试用例编写、测试实施，根据项目计划，测试分成哪些测试阶段（如单元测试、集成测试、系统测试阶段，α、β 测试阶段等），每个阶段的工作重点以及投入资源等。

6. 其他

测试计划还要包括测试计划编写的日期、作者等信息。

3.3 测试设计

根据测试计划设计测试方案，测试设计过程输出的是各测试阶段使用的测试用例，为每一个测试需求确定测试用例集，并且确定执行测试用例的测试过程。根据软件测试计划、软件需求、软件构架设计、软件详细设计等文档内容，设计测试用例具体如下：

(1) 对每一个测试需求，确定其需要的测试用例。

(2) 对每一个测试用例，确定其输入及预期结果。

(3) 确定测试用例的测试环境配置、需要的驱动程序。

(4) 编写测试用例文档。

(5) 对测试用例进行同行评审。

测试执行按以下步骤进行，即单元测试、集成测试、确认测试、系统测试和验收测试，如图3.2所示。

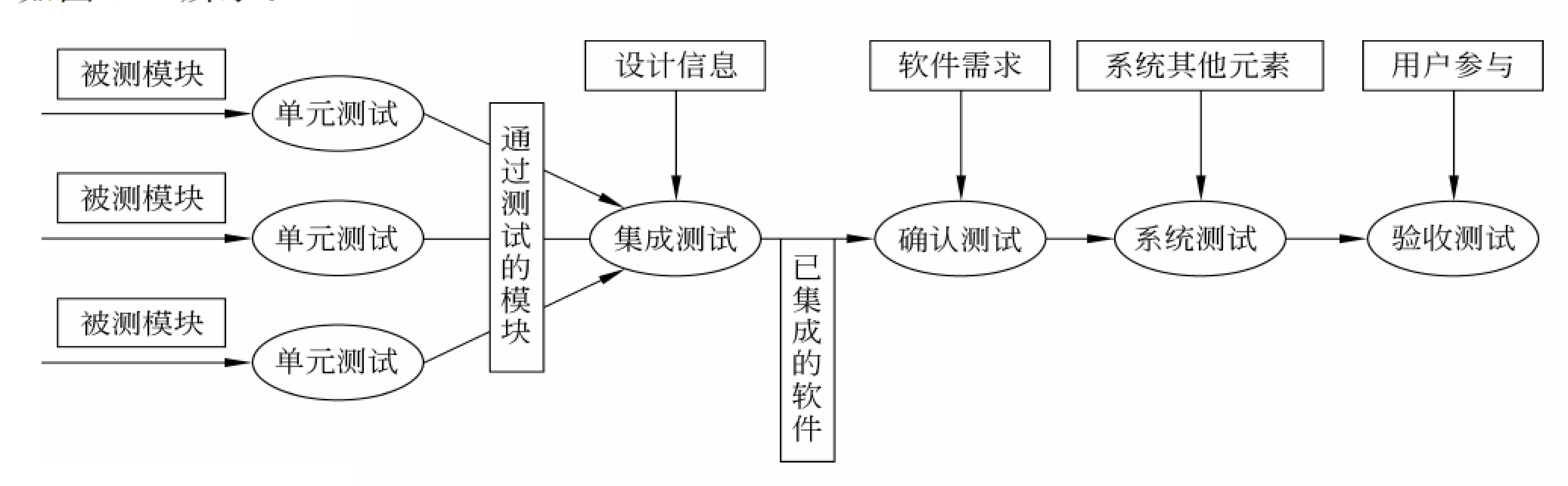


图3.2 软件测试执行过程

(1) 单元测试：通过对每个最小的软件模块进行测试，对源代码的每一个程序单元实行测试，检查各个程序模块是否正确地实现了规定的功能，确保其能正常工作。

(2) 集成测试：对已测试过的模块进行组装集成，目的在于检验与软件设计相关的程序结构问题。

(3) 确认测试：检验软件是否满足需求规格说明中的功能和性能需求，确定软件配置完全、正确。

(4) 系统测试：检验软件产品能否与实际运行环境中整个系统的其他部分(如硬件、数据库及操作人员)协调工作。

(5) 验收测试：作为检验软件产品质量的最后一道工序，主要让用户对软件进行测试。并将重新执行已经做过的测试的某个子集，保证没有引入新的错误。

3.4 单元测试

3.4.1 概述

单元测试是在软件开发过程中进行的最低级别的测试活动，其测试的对象是软件设

计的最小单位。在传统的结构化编程语言中,如 C 语言,单元测试的对象一般是函数或子过程。面向对象的语言中,单元测试的对象可以是类,或类的成员函数。对于 Ada 语言,单元测试可以在独立的过程和函数上进行,也可以在 Ada 包的级别上进行。第四代语言(4GL)中,这时单元被典型地定义为一个菜单或显示界面。

单元测试一般由开发设计人员本身完成,由编写该单元的开发者设计测试用例,测试该单元并修改缺陷。单元测试用于判断一小段代码的某个特定条件(或者场景)下某个特定函数的行为,主要测试软件设计的最小单元(模块)在语法、格式和逻辑等方面的错误,是否符合功能性等需求,程序的多个模块单元可以并行地进行测试工作。

单元测试一般有如下优点:

(1) 单元测试是验证行为。程序中的每一项功能通过测试来验证其正确性,为其后代码的重构提供了保障。

(2) 单元测试是设计行为。通常在软件的设计阶段,设计人员更多地考虑如何实现软件的某项功能、用户界面等,而不考虑这些实现所包含的代码实例。而单元测试的引入,更多地关注于软件的具体功能实现是否符合需求设计,而不仅仅定位于代码的实现运作机制上。这种分析设计的过程,可以催生出设计优良和结构紧凑的代码。

3.4.2 内容

单元测试针对程序模块进行测试,主要有以下 5 个任务——模块接口、局部数据结构、边界条件、独立的路径和错误处理,如图 3.3 所示。

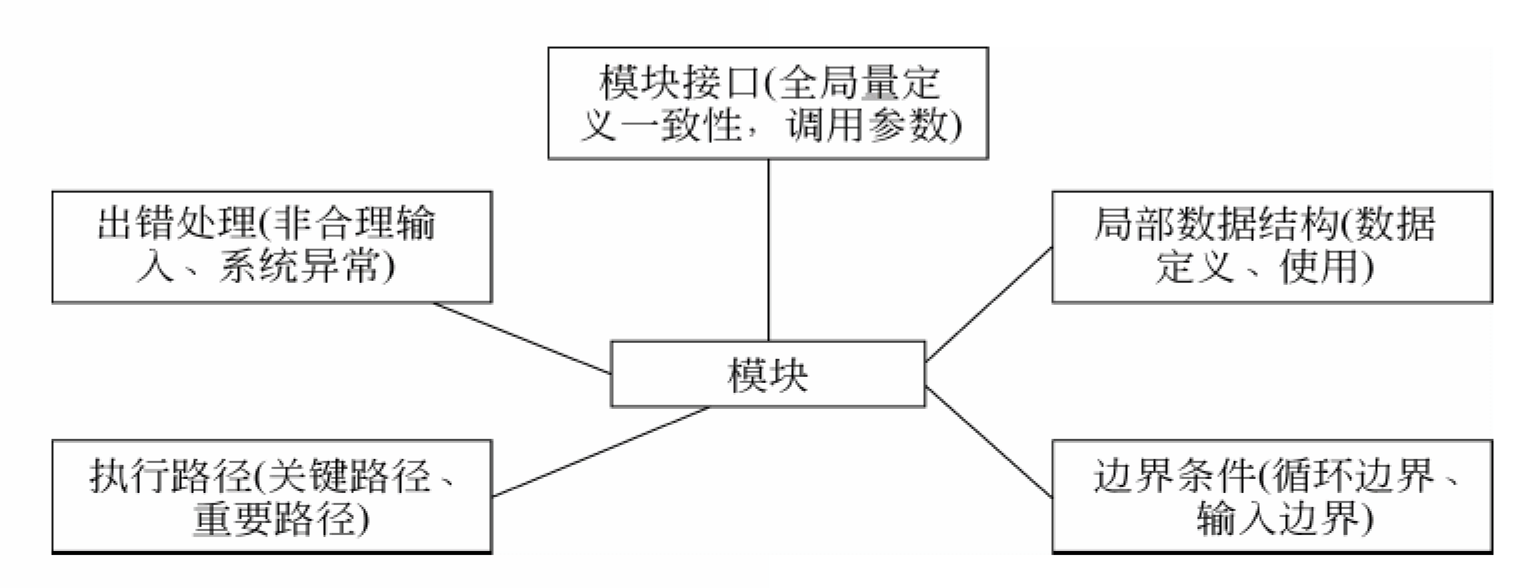


图 3.3 单元测试解决的任务

1. 接口测试

通过对被测模块的数据流进行测试,检查进出模块的数据是否正确。因此,必须对模块接口,包括参数表、调用子模块的参数、全程数据、文件输入输出操作进行测试,涉及的内容如下。

(1) 模块接受输入的实际参数个数与模块的形式参数个数是否一致。

(2) 输入的实际参数与模块的形式参数的类型是否匹配。

(3) 输入的实际参数与模块的形式参数所使用单位是否一致。

(4) 调用其他模块时,所传送的实际参数个数与被调用模块的形式参数的个数是否相同。

(5) 调用其他模块时,所传送的实际参数与被调用模块的形式参数的类型是否匹配。
(6) 调用其他模块时,所传送的实际参数与被调用模块的形式参数的单位一致。
(7) 调用内部函数时,参数的个数、属性和次序是否正确。
(8) 在模块有多个入口的情况下,是否有引用与当前入口无关的参数。
(9) 是否修改了只读型参数。
(10) 全局变量是否在所有引用它们的模块中都有相同的定义。
如果模块内包括外部 I/O,还应该考虑下列因素:
(1) 文件属性是否正确。
(2) OPEN 与 CLOSE 语句是否正确。
(3) 缓冲区容量与记录长度是否匹配。
(4) 在进行读写操作之前是否打开了文件。
(5) 在结束文件处理时是否关闭了文件。
(6) 正文书写/输入错误。
(7) I/O 错误是否检查并做了处理。

2. 模块局部数据结构测试

测试用例检查局部数据结构的完整性,如数据类型说明、初始化、默认值等方面的问题,并测试全局数据对模块的影响。

(1) 在模块工作过程中,必须测试模块内部的数据能否保持完整性,包括内部数据的内容、形式及相互关系不发生错误。

(2) 局部数据结构应注意以下几类错误:不正确的或不一致的类型说明;错误的初始化或默认值;错误的变量名,如拼写错误或书写错误;下溢、上溢或者地址错误。

3. 模块中所有执行路径测试

测试用例对模块中重要的执行路径进行测试,其中对基本执行路径和循环进行测试往往可以发现大量路径错误。测试用例必须能够发现由于计算错误、不正确的判定或不正常的控制流而产生的错误。

(1) 常见的错误:误解的或不正确的算术优先级、混合模式的运算、错误的初始化、精确度不够精确、表达式的不正确符号表示。

(2) 针对判定和条件覆盖,测试用例能够发现的错误有:不同数据类型的比较;不正确的逻辑操作或优先级;应当相等的地方由于精确度的错误而不能相等;不正确的判定或不正确的变量;不正确的或不存在的循环终止;当遇到分支循环时不能退出;不适当地修改循环变量。

4. 各种错误处理测试

检查模块的错误处理功能是否包含有错误或缺陷。例如,是否拒绝不合理的输入;出错的描述是否难以理解、是否对错误定位有误、是否出错原因报告有误、是否对错误条件的处理不正确;在对错误处理之前错误条件是否已经引起系统的干预等。

(1) 测试出错处理的重点是模块在工作中发生了错误,其中的出错处理设施是否

有效。

(2) 检验程序中的出错处理可能面对的情况有:

- 对运行发生的错误描述难以理解。
- 所报告的错误与实际遇到的错误不一致。
- 出错后,在错误处理之前就引起系统的干预。
- 例外条件的处理不正确。
- 提供的错误信息不足,以至于无法找到错误的原因。

5. 边界条件测试

(1) 边界测试是单元测试的最后一步,必须采用边界值分析方法来设计测试用例,在为限制数据处理而设置的边界处,测试模块是否能够正常工作。

(2) 一些与边界有关的数据类型,如数值、字符、位置、数量、尺寸等特征。

(3) 在边界条件测试中,应设计测试用例检查以下情况:

- 在 n 次循环的第 0 次、1 次、n 次是否有错误。
- 运算或判断中取最大值、最小值时是否有错误。
- 数据流、控制流中刚好等于、大于、小于确定的比较值是否出现错误。

3.4.3 步骤

在源程序代码编制完成,经过评审和验证,确认没有语法错误之后,开始设计单元测试的测试用例。由于模块并不是一个独立的程序,测试模块时应考虑它和外界的联系,因此使用一些辅助模块去模拟与被测模块相关的外界模块。辅助模块分为驱动模块和桩模块两种。

1. 驱动模块

驱动模块用来模拟被测试模块的上一级模块,相当于被测模块的主程序,用于接收测试数据,并把这些数据传送给被测模块,启动被测模块,最后输出实测结果。

2. 桩模块

桩模块用来模拟被测模块工作过程中所调用的模块。桩模块一般只进行很少的数据处理,不需要把子模块所有功能都带进来。

被测模块、驱动模块及桩模块共同构成了一个测试环境,如图 3.4 所示。

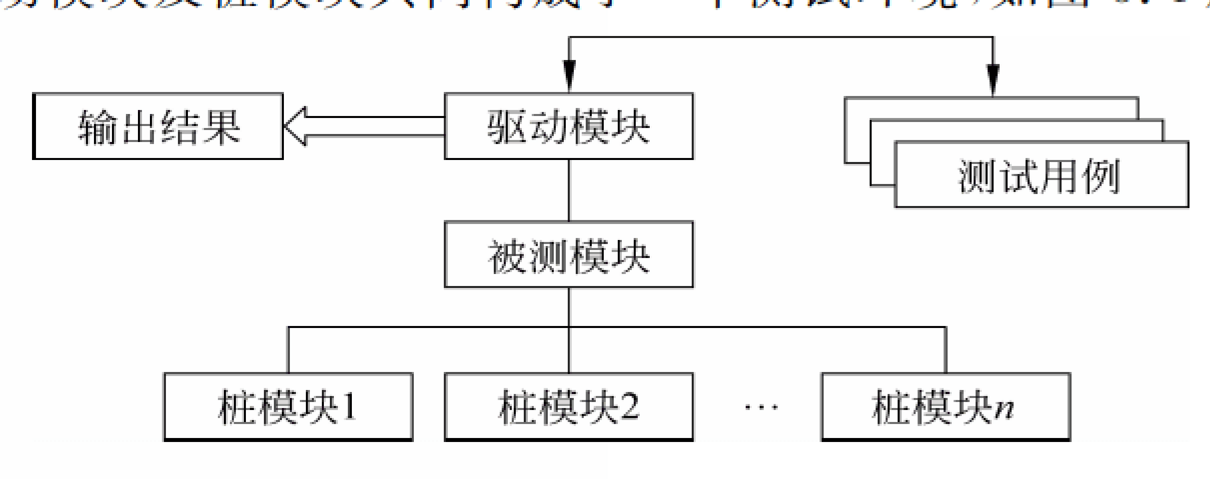


图 3.4 单元测试的测试环境

3.5 集成测试

经过单元测试之后，每个模块都能单独工作，但这些模块组装之后却往往不能正常工作，这是由于以下原由导致。

(1) 模块相互调用时引入了新的问题，例如数据可能丢失，模块之间的相互影响。

(2) 子模块分别实现了子功能，但组合后无法实现主功能。

(3) 子模块所产生的误差由于模块的组合，不断的积累导致错误的产生。

(4) 全局数据结构与局部数据结构的重复出现等错误。

因此，单元测试之后需要进行集成测试。集成测试又名组装测试，是根据模块之间的依赖接口的关系图进行的测试。

集成测试是在开发环境，或是一个独立的测试环境下进行的，由一个独立测试观察员来监控测试工作。

3.5.1 主要任务

集成测试是组装软件的系统测试技术之一，按设计要求把通过单元测试的各个模块组装在一起之后，要求软件系统符合实际软件结构，发现与接口有关的各种错误。集成测试主要适应于如下几种软件系统。

(1) 对软件质量要求较高的软件系统，如航天软件、电信软件、系统底层软件等都必须做集成测试。

(2) 使用范围比较广，用户群数量较大的软件必须做集成测试。

(3) 使用类似 C/C++ 带有指针的程序语言开发的软件一般必须做集成测试。

(4) 类库、中间件等产品必须做集成测试。

集成测试的主要任务是解决以下 5 个问题。

(1) 将各模块连接起来，检查模块相互调用时，数据经过接口是否丢失。

(2) 将各个子功能组合起来，检查能否达到预期要求的各项功能。

(3) 一个模块的功能是否会对另一个模块的功能产生不利的影响。

(4) 全局数据结构是否有问题，会不会被异常修改。

(5) 单个模块的误差积累起来，是否被放大，从而达到不可接受的程度。

3.5.2 集成测试方法

集成测试主要测试软件的结构问题，因此测试建立在模块接口上，多为黑盒测试，适当辅以白盒测试。执行集成测试应遵循如下步骤。

① 确认组成一个完整系统的模块之间的关系。

② 评审模块之间的交互和通信需求，确认出模块间的接口。

③ 生成一套测试用例。

④ 采用增量式测试，依次将模块加入到系统，并测试，这个过程以一个逻辑/功能顺序重复进行。

集成测试过程中尤其要注意关键模块测试，关键模块一般具有如下一个或多个特征：同时对应几条需求功能；具有高层控制功能；复杂且易出错；有特殊的性能要求。

集成测试具有非增量式集成和增量式集成以及核心先行集成等测试方法。

1. 非增量式测试方法

非增量式测试方法又名大棒集成方法，采用一步到位的方法来进行测试，对所有模块进行个别的单元测试后，按程序结构图将各模块连接起来，把连接后的程序当作一个整体进行测试。

2. 增量式测试方法

增量式测试方法是指测试从一个模块开始，测试一个模块后，在添加一个模块进行测试。具有有自顶向下、自底向上以及三明治集成测试方法。

1）自顶向下增量式

自顶向下增量式测试按结构图自上而下进行逐步集成和逐步测试。模块集成的顺序是首先集成主控模块（主程序），然后按照软件控制层次结构向下进行集成。自顶向下的集成方式可以采用深度优先策略和广度优先策略。

图 3.5，深度优先的顺序为：T1→T2→T5→T8→T6→T3→T7→T4；而广度优先的顺序是：T1→T2→T3→T4→T5→T6→T7→T8。

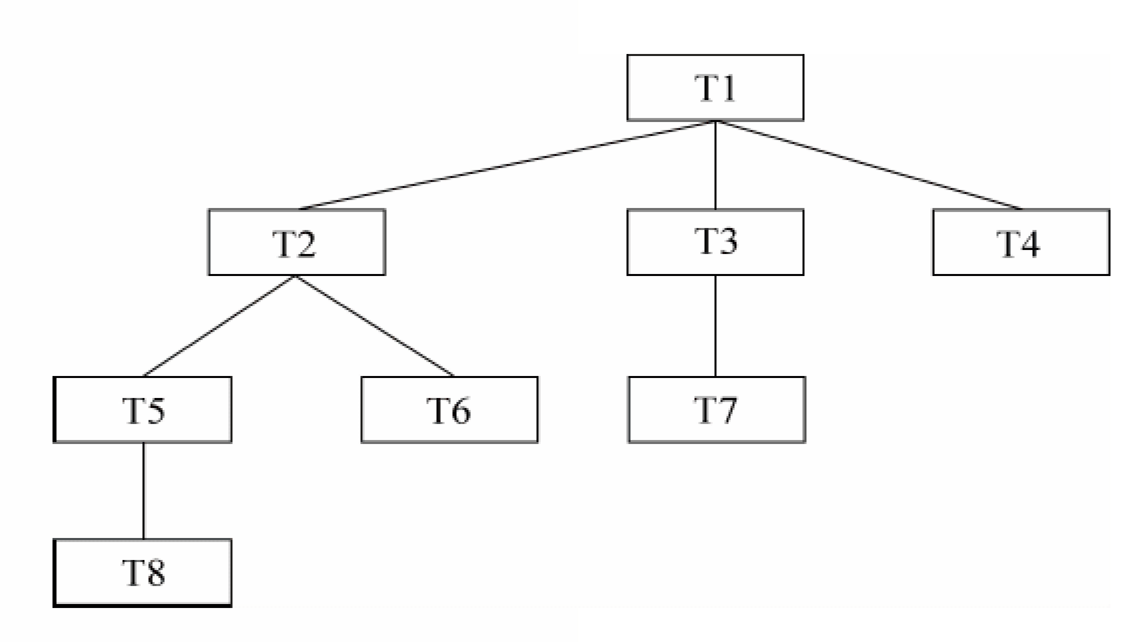


图 3.5　自顶向下增量式示意图

具体步骤如下所示。

① 以主模块为所测试模块兼驱动模块，而所有直属于主模块下属模块全部用桩模块替换，并对主模块进行测试。

② 采用深度优先或广度优先测试方式，用实际模块替换相应桩模块，再用桩代替它们的直接下属模块，从而与已经测试的模块或子系统组装成新的子系统。

③ 进行回归测试排除组装过程中的错误可能性。

④ 判断是否所有的模块都已经组装到了系统中。如果是，结束测试，否则转到步骤②执行。

自顶向下增量式测试方式在测试过程中较早地验证主要的控制点，最多只需要一个驱动模块就可进行测试。当然，随着底层模块的不断增加，会导致底层模块的测试不充

分,特别是被重用的模块。由于每次组装都必须提供桩模块,会使得桩的数目急剧增加,从而维护桩的成本也会快速上升。因此,该方法适合大部分采用结构化编程方法,而且软件的结构相对比较简单。

2）自底向上增量式

自底向上增量式测试是从“原子”模块(软件结构中最低层的模块)开始,按结构图自下而上逐步进行集成和测试,不需要桩模块。图3.6表示了采用自底向上增量式测试的过程,从小族逐步组装成大族进行测试。

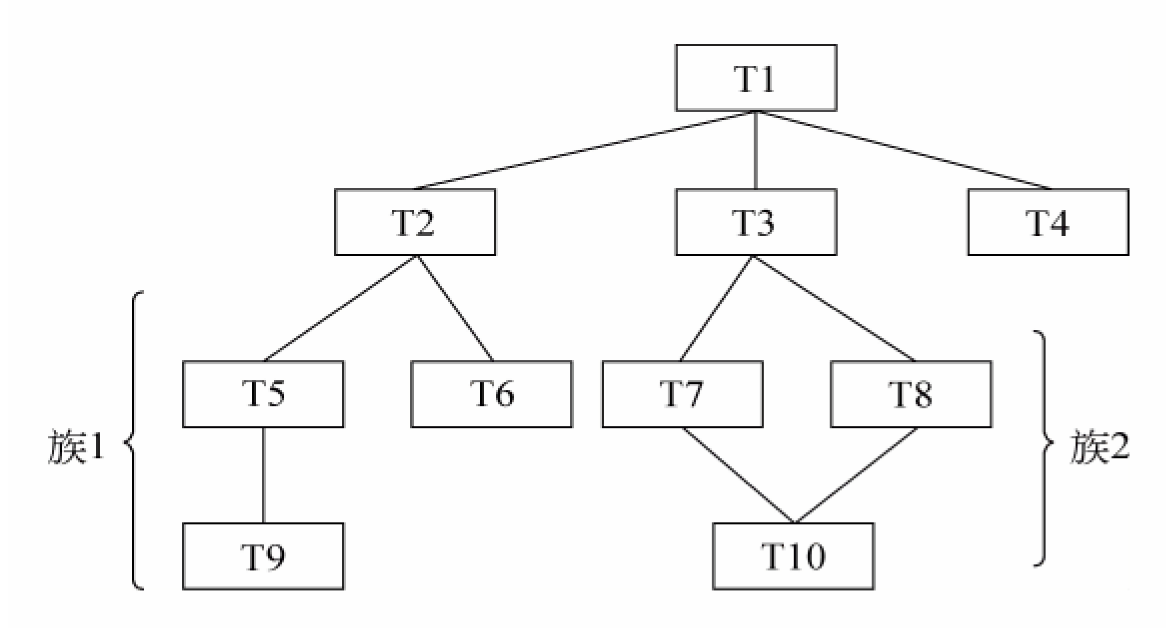


图3.6　自底向上增量式示意图

该方法具体实现由下列几个步骤完成。

① 把低层模块组合成实现某个特定的软件子功能的族。

② 写一个驱动程序(用于测试的控制程序),协调测试数据的输入和输出。

③ 对由模块组成的子功能族进行测试。

④ 去掉驱动程序,沿软件结构由下向上移动,把子功能族组合成更大的功能族。

⑤ 按步骤②～步骤④不断重复上述过程,直到完成。

虽然模拟中断或异常需要设计一定的桩模块,总体上减少了桩模块的工作量。在测试初期,可以并行进行集成,相应地比使用自顶向下的方式效率高。随着集成到顶层,整个系统变得越来越复杂,对于底层的一些模块将很难覆盖。

3）三明治集成

三明治集成也称混合集成,自顶向下和自底向上的缺点和优点集于一身。三明治集成是把系统分为三层,中间一层为目标层。测试时对目标层上面的一层采用自顶向下的集成测试方式,而对目标层下面的一层使用自底向上的集成策略,最后对目标层进行测试。

表3.1给出了各集成测试策略的分析和对比。

表3.1　集成测试方法的比较

名　　称	自顶向下增量式	自底向上增量式	三明治集成
集成	早	早	早
基本程序工作时间	早	晚	早
需要驱动程序	否	是	是

续表

名　　称	自顶向下增量式	自底向上增量式	三明治集成
需要桩程序	是	否	是
工作并行性	低	中	中
特殊路径测试	难	容易	中等
计划与控制	难	容易	难

3. 核心先行集成

核心先行集成测试首先保证一些重要功能和服务的实现测试，对于快速软件开发有效。采用此种模式的测试，要求系统一般应能明确区分核心软件部件和外围软件部件，借助于自动化工具进行高频度的集成测试。

下面给出非增量式集成和增量式集成比较结果。

(1) 非增量式集成测试模式是先分散测试，然后集中起来集成测试。如果在模块的接口处存在错误，只会在最后的集成测试时一下子暴露出来。非增量式集成测试时可能发现很多错误，但为每个错误定位和纠正非常困难，并且在改正一个错误的同时又可能引入新的错误，从而更难断定出错的原因和位置。因此，非增量式集成测试模式只能适合在规模较小的应用系统中使用。与此相反，增量式集成测试采用逐步集成和逐步测试的方法，测试的范围逐步增大，从而错误易于定位和纠正。因此，增量式集成测试比非增量式集成测试员有比较明显的优越性。

(2) 自顶向下测试的主要优点在于自然地做到逐步求精，从一开始让测试者了解系统的框架。它的主要缺点是需要提供驱动模块，而驱动模块可能不能反映真实情况，因此测试有可能不充分。

(3) 自底向上测试的优点在于，由于驱动模块模拟了所有调用参数，从而测试数据没有困难。其主要缺点在于，只有到最后一个模块被加入之后才能知道整个系统的框架。

(4) 三明治集成测试采用自顶向下、自底向上集成相结合的方式，并采取持续集成策略，有助于尽早发现缺陷，有利于提高工作效率。

总之，采用自顶向下集成测试和自底向上的集成测试方案较为常见。在实际测试工作中，应该结合项目的实际环境及各测试方案适用的范围进行合理的选型。

3.6　确认测试

确认测试又称合格性测试，用于验证软件的有效性，即验证软件的功能和性能及其他特性是否与用户的要求一致。确认测试阶段所做工作如图 3.7 所示，进行有效性测试以及软件配置复审，通过专家鉴定之后，才能成为可交付的软件。

有效性测试是在模拟的环境下，运用黑盒测试的方法，验证被测软件是否满足需求规

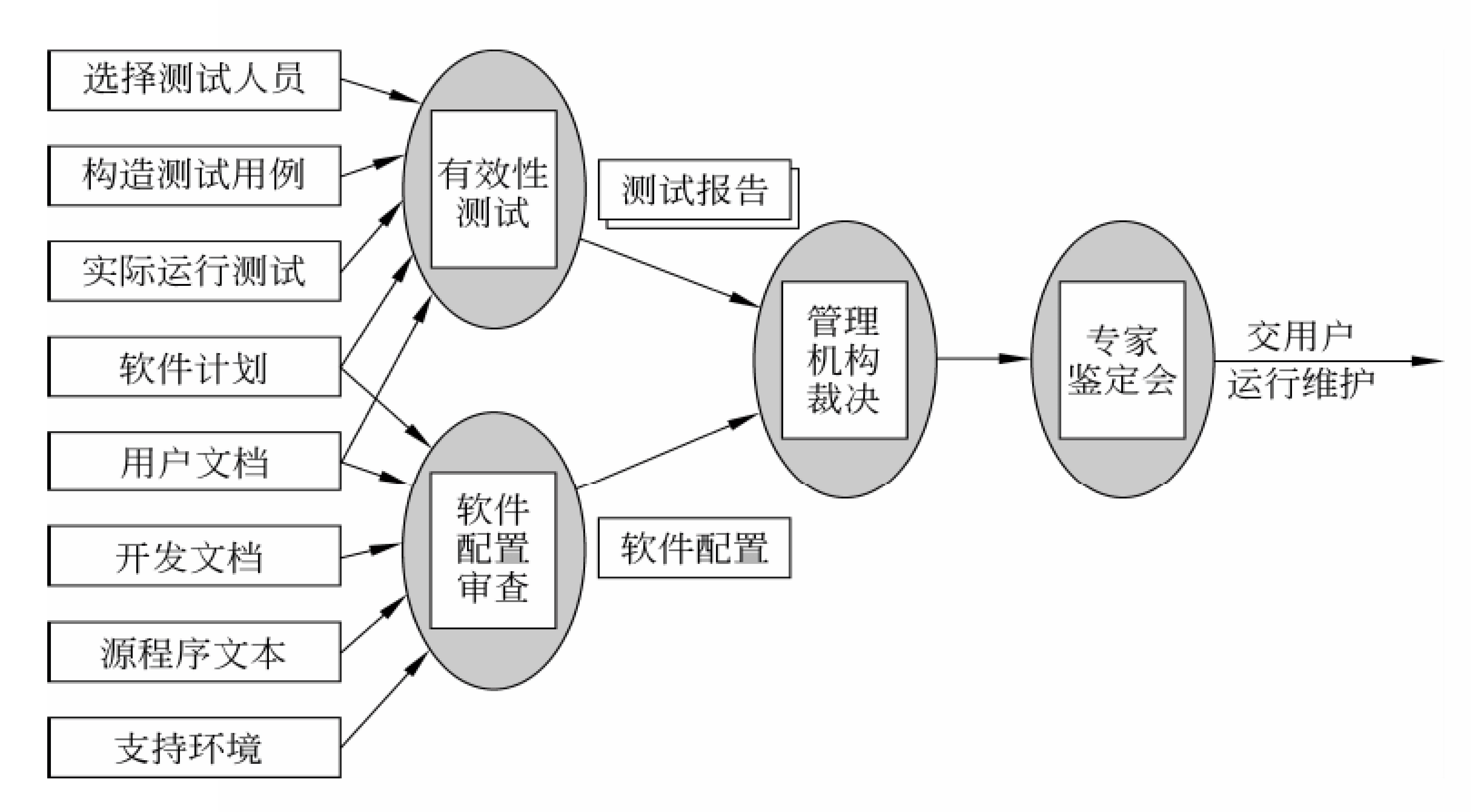


图 3.7　确认测试的步骤

格说明书列出的需求。为此，需要制定测试计划，规定要做测试的种类，制定一组测试步骤，描述具体的测试用例。通过实施预定的测试计划和测试步骤，确定软件的特性是否与需求相符，确保所有的软件功能需求都能得到满足，所有的软件性能需求都能达到，所有的文档都是正确的且便于使用。

软件配置复查的目的是保证软件配置的所有成分，包括与实际运行环境中整个系统的支持环境都应齐全，各方面的质量都符合要求。在确认测试的过程中，应当严格遵守用户手册和操作手册中规定的使用步骤，以便检查这些文档资料的完整性和正确性，记录发现的遗漏和错误，并且适当地补充和改正。

3.7　系统测试

以需求规格说明书作为依据，将作为整个软件系统与计算机硬件、外设、支持软件、数据和人员等其他系统元素结合起来，在实际运行（使用）环境下，对计算机系统进行的测试。系统测试完全采用黑盒测试技术，因为已不需要考虑组件模块的实现细节，而主要是根据需求分析时确定的标准检验软件是否满足功能、行为、性能和系统协调性等方面的要求。系统测试的目标不是要找出软件故障，而是要证明系统的性能。

系统测试停止的条件如下：

（1）系统测试用例设计已经通过评审。

（2）按照系统测试计划完成了系统测试。

（3）达到了测试计划中关于系统测试所规定的覆盖率的要求。

（4）被测试的系统每千行代码必须发现 1 个错误。

（5）系统满足需求规格说明书的要求。

（6）在系统测试中发现的错误已经得到修改，各级缺陷修复率达到标准。

3.8 验收测试

验收测试以用户为主的测试，软件开发人员和质量保证人员也应参加。由用户参加设计测试用例，通过用户界面输入测试数据，分析测试的输出结果。一般使用生产中的实际数据进行测试。在测试过程中，除了考虑软件的功能和性能外，还应对软件的可移植性、兼容性、可维护性、错误的恢复功能等进行确认。

3.8.1 α 测试和 β 测试

在软件交付使用之后，用户在使用过程中常常会发生各种问题。如对软件操作使用方法的误解、异常的数据组合等。α 测试和 β 测试用于发现可能只有最终用户才能发现的错误。α 测试是在开发环境下或者公司内部的用户在模拟实际操作环境下，由用户参与的测试。其测试目的主要是评价软件产品的功能、可使用性、可靠性、性能等，特别是对于软件的界面和使用方式的测试。

β 测试是在实际使用环境下进行的测试。与 α 测试不同，开发者通常不在测试现场。在 β 测试中，由用户记下遇到的所有问题，包括真实的以及主观认定的，定期向开发者报告，开发者在综合用户的报告之后做出修改，最后将软件产品交付给全体用户使用。β 测试着重于产品的支持性，包括文档、客户培训和支持产品生产能力。只有当 α 测试达到一定的可靠程度时，才能开始 β 测试。

α 测试和 β 测试过程如图 3.8 所示。

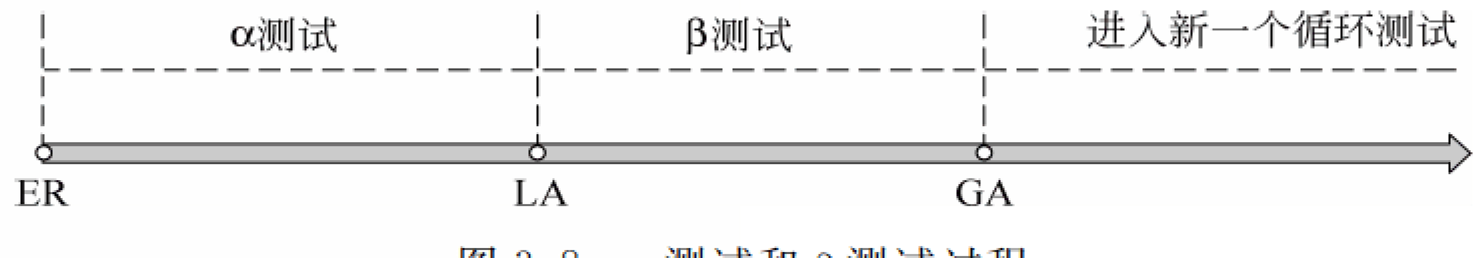


图 3.8　α 测试和 β 测试过程

当软件通过最后阶段的测试—验收测试或质量全面评估测试，从研发阶段来看，ER (Engineering Release，工程发布）作为一个里程碑，随后将软件推向市场。进行 α 测试之后，到达了 LA（Limited Available，有限可用）里程碑，LA 是指由于测试覆盖率不能 100%，软件功能并不能全部使用。LA 之后所发现的缺陷，通过 β 测试，到达 GA (General Available，全面可用）里程碑，此时所有功能全部可以使用。

3.8.2 回归测试

软件生命周期中的任何一个阶段发生了改变，就可能给软件带来缺陷。回归测试是一种验证已变更系统的完整性与正确性的测试技术，是指重新执行已经做过的测试的某个子集，以保证修改没有引入新的错误或者发现由于更改而引起的之前为发现的错误，也就是保证改变没有带来非预期的副作用。因此，软件开发的各个阶段会进行多次回归测试。

1. 回归测试实施前提

（1）当软件中所含错误被发现时，如果错误跟踪与管理系统不够完善，可能会遗漏对这些错误的修改。

（2）开发者对错误理解的不够透彻，也可能导致所做的修改只修正了错误的外在表现，而没有修复错误本身，从而造成修改失败。

（3）修改还有可能产生副作用，从而导致软件未被修改的部分产生新的问题，使本来工作正常的功能产生错误。

微软公司测试经验表明，一般修复三到四个错误会产生一个新的错误。同样，新代码加入软件的时候，除了新代码有可能含有错误外，还有可能对原有的代码带来影响。因此，软件一旦发生变化，必须重新补充新的测试用例，测试软件功能，确定修改是否达到预期目的，检查修改是否损害原有功能。

2. 回归测试用例的选择

回归测试用例的选择主要采用如下几种方法。

1）选择全部测试用例

选择测试用例库中的所有测试用例作为回归测试用例，这是一个较为保险的方法。

2）基于风险选择测试用例

基于一定的风险标准从测试用例库中选择部分测试用例形成测试包而进行的测试。按测试优先级来来选择最重要的、关键的和可疑的测试，而跳过那些非关键的、优先级别低的或者高稳定的测试，从而使得测试工作量会大为减轻。

3）基于操作剖面选择测试用例

由于回归测试时可以优先选择那些针对最重要或最频繁使用功能的测试用例，释放和缓解最高级别的风险，有助于尽早发现那些对可靠性有最大影响的故障。

4）再测试修改部分

这种方式是基于开发对修改的影响区域有较大把握时所采取的一个策略。通过相关性分析软件的修改情况及其影响，将回归测试局限于被改变的模块和它的接口上，此时只选择相应的测试用例来做回归测试。此策略风险最大，但成本也是最低的，通常用于做小回归测试。

以上四种回归测试策略各有优缺点，实际应用中应根据项目的资源：进度及项目开发的模式等实际情况来选择最优策略。

3. 回归测试的两个策略

回归测试贯穿整个测试的各个阶段，其目的是检验已经被发现的缺陷有没有被正确地修改和修改过程中有没有引发新的缺陷，可以采用如下的策略进行回归测试。

1）完全重复测试

完全重复测试是指将所有的测试用例，全部再完整地执行一遍，以确认问题修改的正

确性和修改后周边模块是否受到影响。由于要把用例全部执行，所以会增加项目成本，也会影响项目进度，很难完全执行。

2）选择性重复测试

选择性重复测试是指可以选择一部分进行执行，以确认问题修改的正确性和修改后周边是否受到影响。下面介绍几种有用的方法。

（1）覆盖修改法。

针对发生错误的模块，选取这个模块的全部用例进行测试．这样只能验证本模块是否还存在缺陷，但不能保证周边与它有联系的模块不会因为这次改动而引入缺陷。这类回归测试仅根据修改的内容来选择测试用例，仅保证修改的缺陷或新增的功能被实现，其效率最高，风险也最大。

（2）周边影响法。

除了把出错模块的用例执行之外，把周边和它有联系的模块的用例也执行一边，保证回归测试的质量，需要分析修改可能影响到那部分代码或功能、对于所有受影响的功能和代码，其对应的所有测试用例都将被回归。如何判断哪些功能或代码受影响，往往依赖于测试人员的经验和开发过程的规范性。

（3）指标达成法。

根据一定的覆盖率指标选择回归测试。例如，规定修改范围内的测试是90％，其他范围内的测试阈值为60％，该方法一般是在相关功能影响范围难以界定时使用。

（4）基于操作剖面。

如果测试用例是基于软件操作剖面开发的，测试用例的分布情况将反映系统的实际使用情况。回归测试所使用的测试用例个数由测试预算确定，可以优先选择针对最重要或最频繁使用功能的测试用例，尽早发现对可靠性有最大影响的故障。

（5）基于风险选择测试。

根据缺陷的严重性来进行测试，基于一定的风险标准从测试用例库中选择回归测试包。选择最重要的、关键的以及可疑的测试，跳过那些次要的、例外的测试用例或功能相对非常稳定的模块。

4. 回归测试的流程

回归测试的流程一般具有如下步骤：

① 在测试策略制定阶段，制定回归测试策略。

② 确定回归测试版本。

③ 回归测试版本发布，按照回归测试策略执行回归测试。

④ 回归测试通过，关闭缺陷跟踪单。

⑤ 回归测试不通过，缺陷单返回开发人员．等重新修改，再次做回归测试。

每当一个新的模块被当作集成测试的一部分加进来的时候，软件就发生了改变。新的数据流路径建立起来，新的I/O操作可能也会出现，还有可能激活了新的控制逻辑。这些改变可能会使原本工作得很正常的功能产生错误。在集成测试策略的环境中，回归

测试是对某些已经进行过的测试的某些子集再重新进行一遍，以保证改变不会传播无法预料的副作用。

5. 回归测试与一般测试比较

回归测试与一般测试相比有其特点，分别从测试计划的可获性、测试范围、时间分配、开发信息、完成时间和执行效率进行介绍。

（1）测试用例的新旧：一般测试根据系统规格说明书和测试计划进行，测试用例都是新的。而回归测试可能是更改了的规格说明书、修改过的程序和需要更新的测试计划，测试用例往往是旧的。

（2）测试范围：一般测试目标是检测整个程序的正确性，而回归测试目标是检测被修改的相关部分正确性以及它与系统原有功能的整合。

（3）时间分配：一般测试所需时间通常是在软件开发之前预算，而回归测试所需的时间（尤其是修正性的回归测试）往往不包含在整个产品进度表。

（4）完成时间：由于回归测试只需测试程序的一部分，完成所需时间通常比一般测试所需时间少。

（5）执行频率：回归测试在一个系统的生命周期内往往要多次进行，一旦系统经过修改就需要进行回归测试。因此，因为测试执行频率远远高于一般测试。

3.9 评估测试

软件评估测试主要目的有两个：一是量化测试进程，判断测试进行的状态，决定什么时候测试可以结束；二是为测试或质量分析报告生成所需的量化数据，如缺陷清除率、测试覆盖率等。

软件测试的主要评测方法有测试覆盖、质量评测和性能评测。测试覆盖是对测试完全程度的评测，由测试需求和测试用例的覆盖或已执行代码的覆盖表示。质量评测是对软件系统的可靠性、稳定性以及性能的评测，对测试结果的评估和对测试过程中确定的变更请求进行分析。性能评测检测软件运行时的性能，如传输的最长时间限制、传输的错误率、计算的精度、相应的时限和恢复时限等。

1. 覆盖评测

测试覆盖率作为产品质量的间接指标，对测试覆盖率的评估就是确定测试执行的完全程度，一般有如下几种指标。语句覆盖率、测试用例执行覆盖率、测试需求覆盖率、功能覆盖率、路径覆盖率、决策覆盖率等。覆盖指标主要回答“测试的完全程度如何”这一问题。最常用的覆盖评测是基于需求的测试覆盖和基于代码的测试覆盖。简而言之，测试覆盖是就需求或代码的设计/实施标准而言的完全程度的任意评测。

2. 质量评测

测试覆盖的评估提供对测试完全程度的评测，在测试过程中已发现缺陷的评估提供

了最佳的软件质量指标。

3. 性能评测

评估测试对象的性能行为时，可以使用多种评测，这些评测侧重于获取与行为相关的数据，如响应时间、计时配置文件、执行流、操作可靠性和限制。

经过了评估测试之后，最终测试要停止，一般有如下 5 类常用终止测试的标准和依据。

标准 1：测试超过了预定时间，则终止测试。

标准 2：执行了所有的测试用例，但并没有发现故障，则终止测试。

标准 3：使用特定的测试用例设计方法作为判断测试停止的基础。

标准 4：给出测试停止的要求，例如发现并修改了 100 个软件故障。

标准 5：根据单位时内查出故障的数量决定是否停止测试。

3.10 习　　题

1. 选择题

（1）软件测试是软件质量保证的重要手段，下述测试中属于软件测试最基础环节的是________。

A. 功能测试　　B. 单元测试　　C. 结构测试　　D. 确认测试

（2）从下列叙述中，能够与软件开发各阶段，如需求分析、设计、编码相对应的软件测试是________。

A. 组装测试、确认测试、单元测试　　B. 单元测试、组装测试、确认测试

C. 单元测试、确认测试、组装测试　　D. 确认测试、组装测试、单元测试

（3）单元测试的测试对象是________。

A. 系统　　B. 程序模块　　C. 模块接口　　D. 系统功能

（4）单元测试时用于代替被调用模块的是________。

A. 桩模块　　B. 通信模块　　C. 驱动模块　　D. 代理模块

（5）下列关于 α 测试的描述中准确的是________。

A. α 测试需要用户代表参加　　B. α 测试不需要用户代表参加

C. α 测试是系统测试的一种　　D. α 测试是验收测试的一种

（6）对于软件的 β 测试，下列描述中正确的是________。

A. β 测试就是在软件公司内部展开的测试，由公司专业的测试人员执行的测试

B. β 测试就是在软件公司内部展开的测试，由公司的非专业测试人员执行的测试

C. β 测试就是在软件公司外部展开的测试，由专业的测试人员执行的测试

D. β 测试就是在软件公司外部展开的测试，由非专业的测试人员执行的测试

2. 简答题

(1) 软件测试的生命周期是如何定义的?
(2) α测试与β测试的区别是什么?
(3) 单元测试是什么?其主要任务是什么?
(4) 集成测试方法有几种?集成测试与单元测试的区别是什么?
(5) 如何理解系统测试?
(6) 回归测试与一般测试有几点不同?分别是什么?



黑 盒 测 试

黑盒测试又称功能测试，用于检测每个功能是否都能正常使用。本章介绍了黑盒测试的基本概念，就等价类划分、边界值分析、决策表、因果图、场景法等测试方法进行了详细的解释。

4.1 概　　述

黑盒测试着眼于程序外部结构，不考虑内部逻辑结构，把程序看作一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，在程序接口进行测试，只检查程序功能是否按照需求规格说明书的规定正常执行，程序是否能接收输入数据而产生正确的输出信息。

黑盒测试是以用户的角度，从输入数据与输出数据的对应关系出发进行测试。如果外部特性本身有问题或规格说明的规定有误，黑盒测试方法就无法发现问题。黑盒测试法注重于测试软件的功能需求，主要试图发现下列几类错误。

- 功能不正确或遗漏。
- 界面错误。
- 数据库访问错误。
- 性能错误。
- 初始化和终止错误等。

从理论上讲，黑盒测试只有采用穷举输入测试，把所有可能的输入都作为测试情况考虑，才能查出程序中所有的错误。

【例 4-1】 任意输入三角形的三边，判断三角形类型。

如图 4.1 所示，输入三角形的三条边 a、b、c，设计测试用例数量。假设在字长为 16 位的计算机上运行，则每个整数可能的取值为 2^{16} 种，则 a、b、c 3 条边的各种可能取值的排列组合就有 $2^{16}\times2^{16}\times2^{16}\approx3\times10^{14}$ 种，执行完所有的测试大约需要执行 3×10^{14} 次，也就是说，大约需要执行 3×10^{14} 次才能做到“穷尽”测试，假设执行 1 次需时 1ms，则执行完所有的测试数据就共需 1 万年。因此，完全测试是不可能的，所以要进行有针对性进行测试，选择有效的测试用例。黑盒测试用例设计方法包括等价类划分法、边界值分析法、错误推测法、因

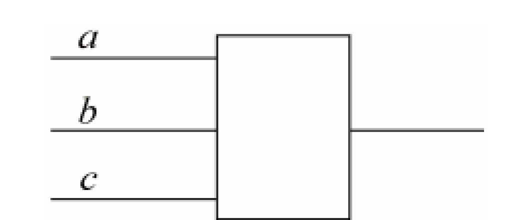


图 4.1　三角形三边取值情况的测试用例

果图法、判定表驱动法、功能图法等。

4.2 等价类划分

等价类是指某个输入域的子集合。在该子集合中，测试某等价类的代表值就等于对该类其他值的测试，对于揭露程序的错误具有等价的效果。因此，全部输入数据合理划分为若干等价类，在每一个等价类中取一个数据作为测试的输入条件，就可以用少量代表性的测试数据取得较好的测试结果。

等价类划分为两种情况：有效等价类和无效等价类。

（1）有效等价类：对于程序的规格说明来说是合理的，有意义的输入数据构成的集合，利用有效等价类可检验程序是否实现了规格说明中所规定的功能和性能。

（2）无效等价类：与有效等价类相反，是指对程序的规格说明无意义、不合理的输入数据构成的集合。

4.2.1 划分原则

按照如下几条规则对等价类进行划分。

1. 按区间划分

如果规定了输入值的范围或值的个数的情况下，通常定义一个有效等价类和两个无效等价类。例如，输入条件规定了 x 是 1～999 的整数。则等价类划分如图 4.2 所示。

1　999
无效等价类 <1　有效等价类　无效等价类 >999

图 4.2　等价类划分举例

2. 按限制条件和规则划分

当规定了输入的规则时，则可以划分出一个有效的等价类（符合规则）和若干无效的等价类（从不同角度违反规则）。例如，C 语言规定，每个语句以“；”结束，则其有效类 1 个、无效类若干（以“，”结束、以“：”结束或以空格结束等）。

3. 按数值集合划分

当输入数据是一组值，而且程序对不同输入值做不同处理，则每个允许的输入值是一个有效等价类，并有一个无效等价类。例如，教工分房方案中，按教授、副教授、讲师、助教分别计分，则有效类 4 个、无效类 1 个。

4. 细分等价类

当处理表格时，有效类可分为空表、含一项的表、含多项的表等。

4.2.2 设计测试用例步骤

等价类设计测试用例一般经历如下步骤：

① 形成等价类表，每一等价类规定一个唯一的编号，如(1)、(2)、(3)等。

② 设计测试用例，使其尽可能多地覆盖尚未覆盖的有效等价类，重复这一步骤，直到所有有效等价类均被测试用例所覆盖。

③ 设计一新测试用例，使其只覆盖一个无效等价类，重复这一步骤直到所有无效等价类均被覆盖(通常，程序执行一个错误后不继续检测其他错误，故每次只测一个无效类)。

【例 4-2】 某城市电话号码由三部分组成。地区码由空白或三位数字组成；前缀是非 0 或 1 开头的三位数字；后缀是四位数字。

【解析】 步骤 1：等价类划分，如表 4.1 所示。

表 4.1 等价类划分

输入条件	有效等价类	无效等价类
地区码	空白(1) 3 位数字(2)	有非数字字符 (5) 少于 3 位数字 (6) 多于 3 位数字 (7)
前缀	从 200 到 999 之间的 3 位数字(3)	有非数字字符 (8) 起始位为‘0’(9) 起始位为‘1’(10) 少于 3 位数字 (11) 多于 3 位数字 (12)
后缀	4 位数字(4)	有非数字字符 (13) 少于 4 位数字 (14) 多于 4 位数字 (15)

步骤 2：确定测试用例。

(1) 对表中 4 个有效等价类，如表 4.2 所示。

表 4.2 有效等价类

测试数据	测试范围	期望结果
( )276-2345 (635)805-9321	等价类(1)(3)(4) 等价类(2)(3)(4)	有效 有效

(2) 对表中 11 个无效等价类应选择 11 个测试用例，如表 4.3 所示。

表 4.3 无效等价类

测试数据	测试范围	期望结果
(20A)123-4567	无效等价类(5)	无效
(33)234-5678	无效等价类(6)	无效
(7777)345-6789	无效等价类(7)	无效
⋮	⋮	⋮
( )276-23456	无效等价类(15)	无效

【例 4-3】 NextDate 函数实现如下功能：输入年月日 3 个变量，分别为 day(日期)、month(月)、year(年)，输出为输入日期的后一天的日期。例如，输入为 1964 年 8 月 3 日，则 NextDate 函数的输出为 1964 年 8 月 4 日。设计测试用例如下所示。

【解析】 有效等价类如下所示：

```
D1={ day :1<=  day <=31 }
M1={month :1<=  month <=12 }
Y1={ year :1912<=year <=2050 }
```

无效等价类如下所示：

```
D2: {  day : day<1}
D3: {  day : day>31}
M2: {  month : month<1}
M3: {  month : month>12}
Y2: {  year : year<1912}
Y3: {  year : year>2050}
```

NextDate 函数的等价类测试用例设计如下：一个有效测试用例使用每个有效等价类中的一个值，无效测试用例中有一个是无效值，其他都取有效值，如表 4.4 所示。

表 4.4 NextDate 函数的等价类测试用例

测试用例	day	month	year	预期输出
Test1	8	8	1998	1998 年 8 月 9 日
Test2	−1	8	1998	D2
Test3	32	8	1998	D3
Test4	8	−1	1998	M2
Test5	8	13	1998	M3
Test6	8	8	1911	Y2
Test7	8	8	2051	Y3

4.3 边界值分析法

实践证明，大量的错误发生在输入或输出范围的边界上，而不是发生在输入输出范围的内部，例如，程序的许多错误出现在数组的下标、循环控制变量等边界附近。因此针对各种边界情况设计测试用例，可以查出更多的错误。边界值分析作为等价类划分方法的补充，是通过选择等价类边界值作为测试用例，而不是选取等价类中的典型值或任意值作为测试数据。

4.3.1 设计原则

边界值分析方法设计测试用例，应确定输入和输出等价类的边界，应当选取正好等于，刚刚大于或刚刚小于边界的值作为测试数据，而不是选取等价类中的典型值或任意值作为测试数据。

边界值分析方法设计测试用例具有如下原则。

(1) 如果输入条件规定了值的范围，则应选取刚达到范围的边界值，以及刚刚超越边界的值作为测试输入数据。例如，输入变量为 X_1、X_2，取值范围是 $a \leqslant X_1 \leqslant b$，$c \leqslant X_2 \leqslant d$，边界分析图如图 4.3 所示。

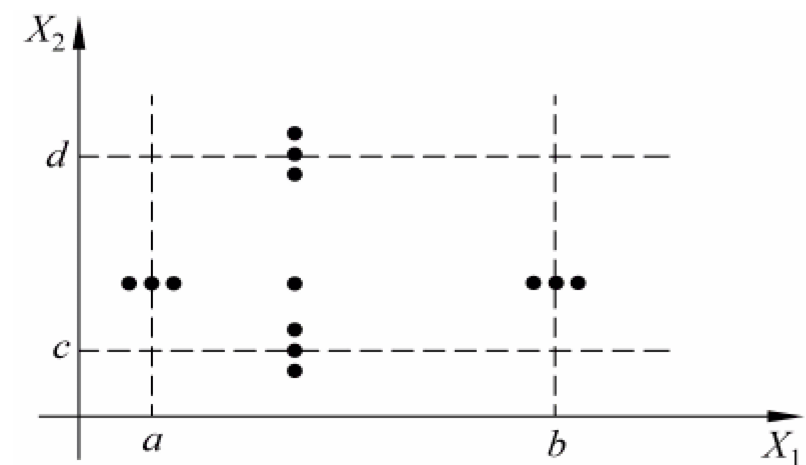


图 4.3 两变量函数边界分析测试用例

(2) 如果输入条件规定了值的个数，则用略低于最小值(Min_-)、最小值(Min)、略高于最小值：(Min_+)、正常值(Normal)、略低于最大值(Max_-)、最大值(Max)、略高于最大值(Max_+)作为测试数据。因此，对于一个含有 n 个变量的程序，保留其中一个变量，其取值为最小值(Min)、略大于最小值(Min_+)、输入值域内的任意值(Normal)、略小于最大值(Max_-)和最大值(Max)，让其余变量取正常值，测试用例数目为 $4 \times n+1$。

(3) 如果程序规格说明给出的输入域或输出域是有序集合，则应选取集合的第一个元素和最后一个元素作为测试用例。

4.3.2 应用举例

【例 4-4】 三角形问题：输入 3 个整数 a、b 和 c 分别作为三角形的 3 条边，通过程序判断由这 3 条边组成的三角形类型是：等边三角形、等腰三角形、一般三角形或非三角形。

【解析】 根据题意，可得到如下三角形 3 边 a、b、c 必须满足如下条件：

- 条件 1：$1 \leqslant a \leqslant 100$；
- 条件 2：$1 \leqslant b \leqslant 100$；
- 条件 3：$1 \leqslant c \leqslant 100$；
- 条件 4：$a < b+c$；
- 条件 5：$b < a+c$；
- 条件 6：$c < b+a$。

如果三角形 3 边 a、b、c 满足条件 1、条件 2、条件 3，则输出下列 4 种情况之一：

(1) 如果不满足条件 4、条件 5、条件 6 中的任何一个，则程序输出为“非三角形”。

(2) 如果有两条边相等，则程序输出为“等腰三角形”。

(3) 如果有三条边相等，则程序输出为“等边三角形”。

(4) 如果三条边都不相等，则程序输出为“一般三角形”。

分析可知，上面 4 种情况相互排斥。由于三角形问题共有 3 个变量，测试用例数目为 $4 \times 3+1=13$ 个。具体情况如表 4.5 所示。

表 4.5　三角形问题测试用例

测试用例	边长 a	边长 b	边长 c	预期输出
Test1	50	50	1	等腰三角形
Test2	50	50	2	等腰三角形
Test3	50	50	50	等边三角形
Test4	50	50	99	等腰三角形
Test5	50	50	100	非三角形
Test6	50	1	50	等腰三角形
Test7	50	2	50	等腰三角形
Test8	50	99	50	等腰三角形
Test9	50	100	50	非三角形
Test10	1	50	50	等腰三角形
Test11	2	50	50	等腰三角形
Test12	99	50	50	等腰三角形
Test13	100	50	50	非三角形

【例 4-5】 某报表处理系统要求用户输入处理报表的日期，日期限制在 2003 年 1 月至 2008 年 12 月，即系统只能对该段期间内的报表进行处理，如日期不在此范围内，则显示输入错误信息。系统日期规定由年、月的 6 位数字字符组成，前四位代表年，后两位代表月。

【解析】 采用边界值分析方法得到结果如表 4.6 所示。

表 4.6　边界值分析法设计测试用例

输 入 条 件	测试用例说明	测试数据
报表日期的类型及长度	1 个数字字符	5
	5 个数字字符	20035
	7 个数字字符	2003005
	有 1 个非数字字符	2003.5
	全部是非数字字符	MAY---
	6 个数字字符	200305
日期范围在有效范围边界上选取数据		200301
		200812
		200300
		200813
月份范围	月份为 1 月	200301

续表

输入条件	测试用例说明	测试数据
	月份为12月	200312
	月份<1	200300
	月份>12	200313

4.4 决策表

决策表又称为判定表，是分析多种逻辑条件下执行不同操作的技术。在程序设计发展的初期，决策表作为程序编写的辅助工具。决策表可以把复杂的逻辑关系和多种条件组合情况表达明确，与高级程序设计语言中的if-else、switch-case等分支结构语句类似，将条件判断与执行的动作联系起来。但与程序语言中的控制语句不同是，决策表能将多个独立的条件和多个动作联系清晰的表示出来。



图4.4 决策表的组成

决策表由四个部分组成，如图4.4所示。

(1) 条件桩：列出了问题得所有条件，通常认为列出的条件次序无关紧要。

(2) 动作桩：列出了问题规定可能采取的操作，这些操作的排列顺序没有约束。

(3) 条件项：列出针对它条件桩的取值，在所有可能情况下的真假值。

(4) 动作项：列出在条件项的各种取值情况下应该采取的动作。

规则：任何条件组合的特定取值及其相应要执行的操作。在决策表中贯穿条件项和动作项的列就是规则。显然，决策表中列出多少条件取值，也就有多少规则，条件项和动作项就有多少列。

所有条件都是逻辑结果(即真/假、是/否、0/1)的决策表称为有限条件决策表，如果条件有多个值，则对应的决策表叫做扩展条目决策表。决策表设计测试用例，条件解释为输入，动作解释为输出。决策表适合以下条件。

(1) if-then-else分支逻辑突出。

(2) 输入变量之间存在逻辑关系。

(3) 涉及输入变量子集的计算。

(4) 输入和输出之间存在因果关系。

(5) 很高的圈复杂度。

如表4.7所示，打印机工作用决策表表示。右上部分的Y表示条件成立，F表示条件不成立，空白表示这个条件成立与否并不影响动作的选择。决策表右下部分中画“√”表示做它左边的相应动作，空白表示不做这项动作。

表 4.7 使用决策表设计打印机的测试用例

条件	不能打印	Y	Y	Y	Y	N	N	N
	红灯闪	Y	Y	N	N	Y	Y	N
	不能识别打印机	Y	N	Y	N	Y	N	Y
动作	检查电源线			√				
	检查打印机数据线	√		√				
	检查是否安装驱动程序	√		√		√		√
	检查墨盒	√	√			√	√	
	检查是否卡纸		√		√			

4.4.1 应用举例

使用判定表设计测试用例的具体步骤如下：

(1) 确定规则的个数。假如有 n 个条件，每个条件有两个取值(0，1)，故有 2^n 种规则。

(2) 列出所有的条件桩和动作桩。

(3) 填入条件项。

(4) 填入动作项，得到初始判定表。

(5) 简化，合并相似规则(相同动作)。

简化就是规则合并有两条或多条规则具有相同的动作，并且其条件项之间存在着极为相似的关系。

① 如图 4.5 所示，其左端，两规则动作项一样，条件项类似，在 1、2 条件项分别取 Y、N 时，无论条件 3 取何值，都执行同一操作。即要执行的动作与条件 3 无关，可合并。“—”表示与取值无关。

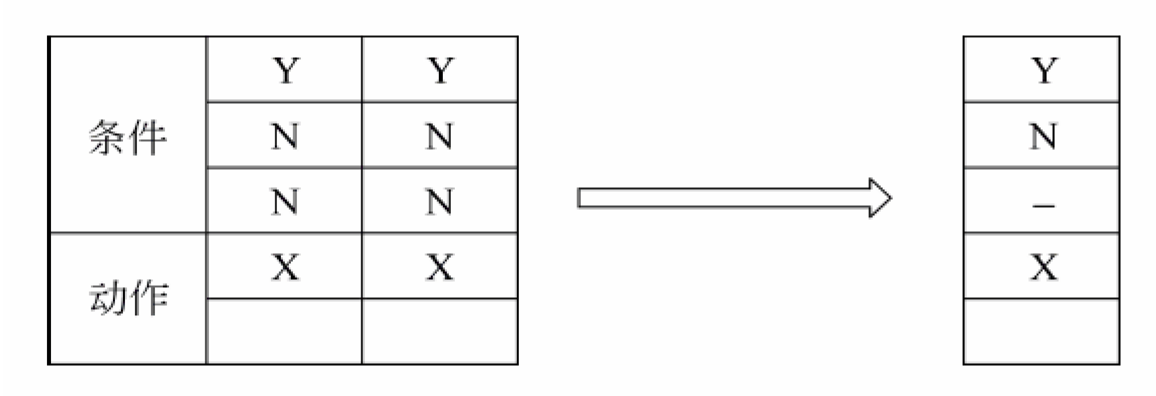


图 4.5 简化规则

② 如图 4.6 所示，无关条件项“—”可包含其他条件项取值，具有相同动作的规则可合并。

【例 4-6】 某国有企业改革重组，对职工重新分配工作的政策是：年龄在 20 岁以下者，初中文化程度脱产学习，高中文化程度当电工；年龄在 20 岁到 40 岁之间者，中学文化程度男性当钳工，女性当车工，大学文化程度都当技术员。年龄在 40 岁以上者，中学文化程度当材料员，大学文化程度当技术员。请用决策表描述上述问题的加工逻辑。

条件	Y	Y
	N	–
	N	N
动作	X	X

⇒

Y
–
N
X

图 4.6　简化规则

【解答】 条件取值表如表 4.8 所示。

表 4.8　条件取值表

条件名	取值	符号	取值数
年龄	≤20	C	M1＝3
	＞20,＜40	D	
	≥40	E	
文化程度	中学	G	M2＝3
	高中	H	
	大学	I	
性别	男	M	M3＝2
	女	F	

根据公式 M1×M2×M3＝3×3×2＝18，由于题意及规则简化，最终结果如表 4.9 所示。

表 4.9　使用决策表设计例 4-4 的测试用例

		1	2	3	4	5	6	7	8	9	10
条件	年龄	C	C	D	D	D	D	D	E	E	E
	文化	G	H	H	G	G	H	I	G	H	I
	性别	—	—	M	M	F	F	—	—	—	—
动作	脱产学习	√									
	电工		√								
	钳工			√	√						
	车工					√	√				
	技术员							√			√
	材料员								√	√	

【例 4-7】 NextDate 函数在【例 4-3】中进行介绍，与【例 4-4】“三角形问题”一样，NextDate 函数对 day、month、year 的无效输入值进行测试。不满足条件 1、条件 2 和条件 3 的任何一个，NextDate 都会产生一个输出，指明相应的变量超出了取值范围。例如，month 的值不在 1～12 之间、day 不在 1～31 之间、year 不在 1912～2050 之间，则

NextDate 函数输出“无效输入日期”。

NextDate 函数问题相当于三角形问题复杂有两点：一是所讨论的输入域的复杂性，month、day、year 三个变量不光各自有取值区间，而且它们之间是有关联的，不是相互独立的；二是确定闰年的规则。

4.4.2 优点和缺点

决策表把复杂问题的各种可能情况一一列出，易于理解。但是，决策表不能表达重复执行动作的缺点。

B. Beizer 指出使用判定表设计测试用例的条件：

(1) 规格说明以判定表形式给出，或很容易转换成判定表。

(2) 条件的排列顺序不会也不影响执行哪些操作。

(3) 规则的排列顺序不会也不影响执行哪些操作。

(4) 每当某一规则的条件已经满足，并确定要执行的操作后，不必检验别的规则。

(5) 如果某一规则得到满足要执行多个操作，这些操作的执行顺序无关紧要。

这 5 个必要条件使得操作的执行完全依赖于条件的组合，对于不满足条件的判定表，可增加其他的测试用例。

4.5 因 果 图

等价类划分法和边界值分析法只是孤立地考虑各个输入数据的测试效果，没有考虑输入数据的组合及其相互制约关系，这样输入条件组合起来可能出错的情况往往被忽视。而输入条件的各种组合数目又是天文数字，因此必须考虑采用一种适合于描述多种条件的组合、产生多个动作的测试用例设计方法，这就是因果图方法。

因果图利用图解法分析输入的各种组合情况，适合于描述多种输入条件的组合、相应产生多个动作的方法。因果图具有如下好处：

(1) 考虑多个输入之间的相互组合、相互制约关系。

(2) 指导测试用例的选择，指出需求规格说明描述中存在问题。

(3) 能够帮助测试人员按照一定的步骤，高效率的开发测试用例。

(4) 因果图法是将自然语言规格说明转化成形式语言规格说明的一种严格的方法，可以指出规格说明存在的不完整性和二义性。

4.5.1 基本术语

下面，介绍因果图的基本图形符号。

1. 原因-结果图

原因-结果图使用了简单的逻辑符号，以直线联接左右结点。左结点表示输入状态(原因)，右结点表示输出状态(结果)。图 4.7 表示规格说明中的 4 种因果关系，其中 c_i 表示原因，通常置于图的左部；e_i 表示结果，通常在图的右部。c_i 和 e_i 均可取值 0 或 1(0

表示某状态不出现，1 表示某状态出现）。

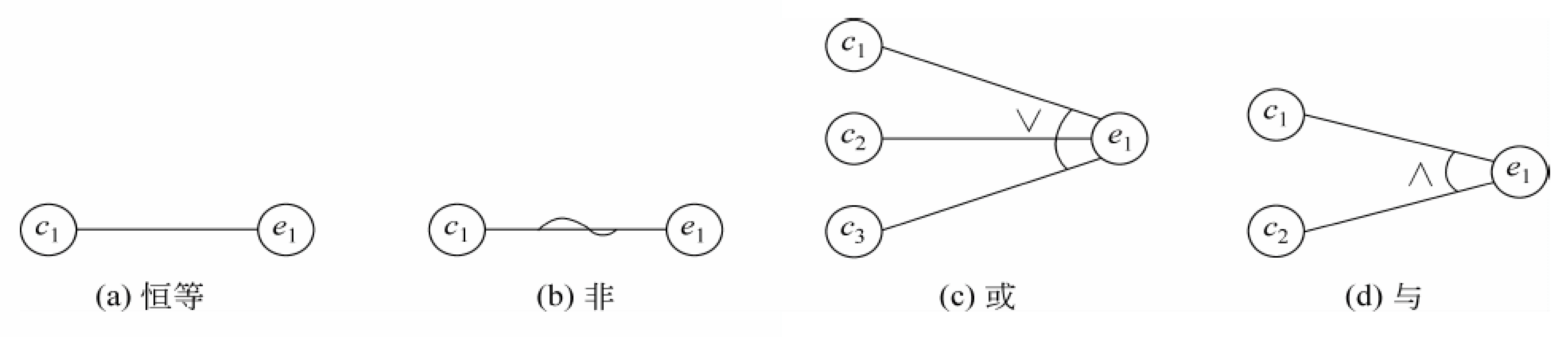


图 4.7　原因—结果图

图 4.7(a)表示“恒等”关系，即若 c_i 是 1，则 e_i 也是 1；否则 e_i 为 0。图 4.7(b)表示“非”关系，即若 c_i 是 1，则 e_i 是 0；否则 e_i 是 1。图 4.7(c)表示“或”关系，可有任意个输入。若 c_1 或 c_2 或 c_3 是 1，则 e_i 是 1；否则 e_i 为 0。图 4.7(d)表示“与”关系，也可有任意个输入。若 c_1 和 c_2 全都是 1，则 e_i 为 1；否则 e_i 为 0。

2. 约束图

输入输出状态相互之间存在的某些依赖关系，称为约束，如图 4.8 所示。

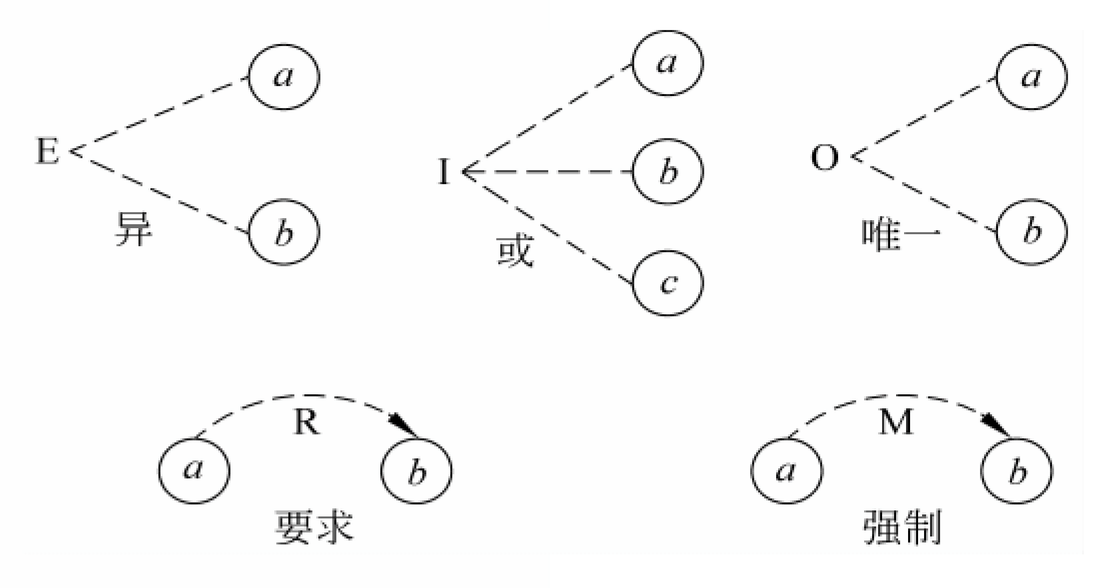


图 4.8　约束图

(1) E 约束(Exclusive，异)：a 和 b 中至多有一个可能为 1，即 a 和 b 不能同时为 1。
(2) I 约束(Inclusive，或)：a、b 和 c 至少有一个是 1，即 a、b 和 c 不能同时为 0。
(3) O 约束(One and Only，唯一)：a 和 b 必须有一个，且仅有一个为 1。
(4) R 约束(Require，要求)：a 是 1 时，结果 b 是 1。
(5) M 约束(Masks，强制)：a 是 1 时，结果 b 是 0。
因果图设计测试用例需要如下步骤，如图 4.9 所示。

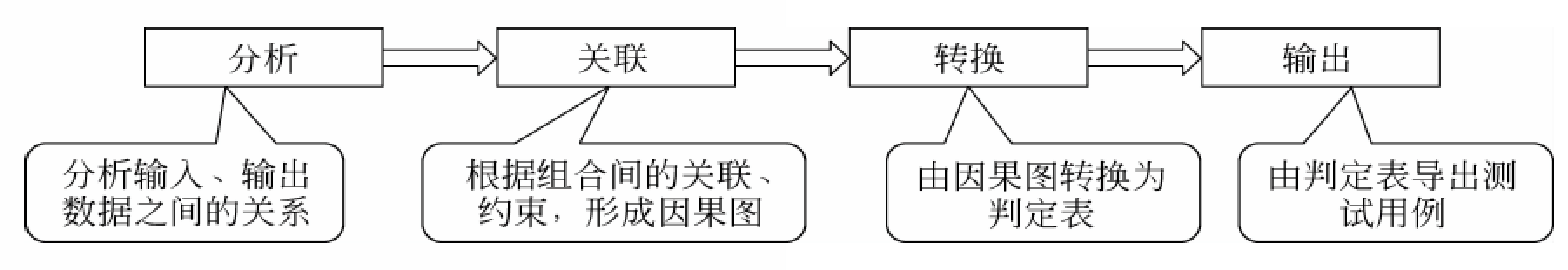


图 4.9　因果图生成测试用例的步骤示意图

① 分析软件规格说明，哪些是原因(即输入条件或输入条件的等价类)，哪些是结果(即输出条件)，给每个原因和结果赋予标识符。

② 分析原因与结果之间，原因与原因之间对应的逻辑关系，用因果图的方式表示出来。

③ 由于语法或环境限制，有些原因与原因之间，原因与结果之间的组合情况不可能出现，在因果图上用一些记号表明这些特殊情况的约束或限制条件。

④ 把因果图转换为判定表。

⑤ 从判定表的每一列产生出测试用例。

对于逻辑结构复杂软件，先用因果图进行图形分析，再用判定表进行统计，最后设计测试用例。当然，对于比较简单的测试对象，可以忽略因果图，直接使用决策表。

4.5.2 应用举例

【例 4-8】 软件需求规格说明如下：第一列字符必须是 A 或 B，第二列字符必须是一个数字，在此情况下进行文件的修改，但如果第一列字符不正确，则给出信息 L；如果第二列字符不是数字，则给出信息 M。

【解答】 采用因果图方法，具体步骤如下所示：

① 分析程序规格说明书，识别哪些是原因，哪些是结果。原因往往是输入条件或者输入条件的等价类，而结果常常是输出条件。

原因：

- 1—第一列字符是 A。
- 2—第一列字符是 B。
- 3—第二列字符是一数字。

结果：

- 21—修改文件。
- 22—给出信息 L。
- 23—给出信息 M。

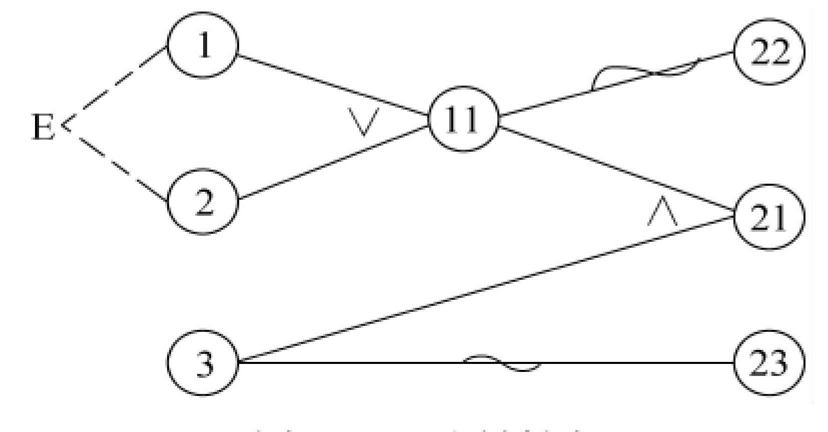


图 4.10 因果图

② 根据原因和结果产生因果图，如图 4.10 所示。

③ 原因 1 和原因 2 不能同时为 1，即第一个字符不可能既是 A 又是 B，有 6 种取值，如表 4.10 所示。

表 4.10 决策表

		1	2	3	4	5	6
原因	1	1	1	0	0	0	0
	2	0	0	1	1	0	0
	3	1	0	1	0	1	0
结果	21	1	0	1	0	0	0
	22	0	0	0	0	1	1
	23	0	1	0	1	0	1
测试用例		A3 A5	AM AN	B5 B4	BN B!	C2 X6	DY P;

4.6 场 景 法

软件系统中流程的控制由事件触发决定，例如，申请项目，需先提交审批单据，再由部门经理审批，通过后由总经理来最终审批，如果审批不通过，则退回。这样，事件不同的触发顺序和处理结果形成事件流，每个事件流触发时的情景便形成了场景。通过运用场景来对系统的功能点或业务流程的描述，可以提高测试效果。场景法一般包含基本流和备用流，从一个流程开始，通过描述经过的路径来确定的过程，经过遍历所有的基本流和备用流来完成整个场景。

4.6.1 基本流和备选流

场景法的描述如图 4.11 所示，图中经过用例的每条路径都用基本流和备选流来表示，直黑线表示基本流，是经过用例的最简单的路径。备选流用不同的色彩表示，一个备选流可能从基本流开始，在某个特定条件下执行，然后重新加入基本流中（如备选流 1 和 3）；也可能起源于另一个备选流（如备选流 2），或者终止用例而不再重新加入到某个流（如备选流 2 和 4）。

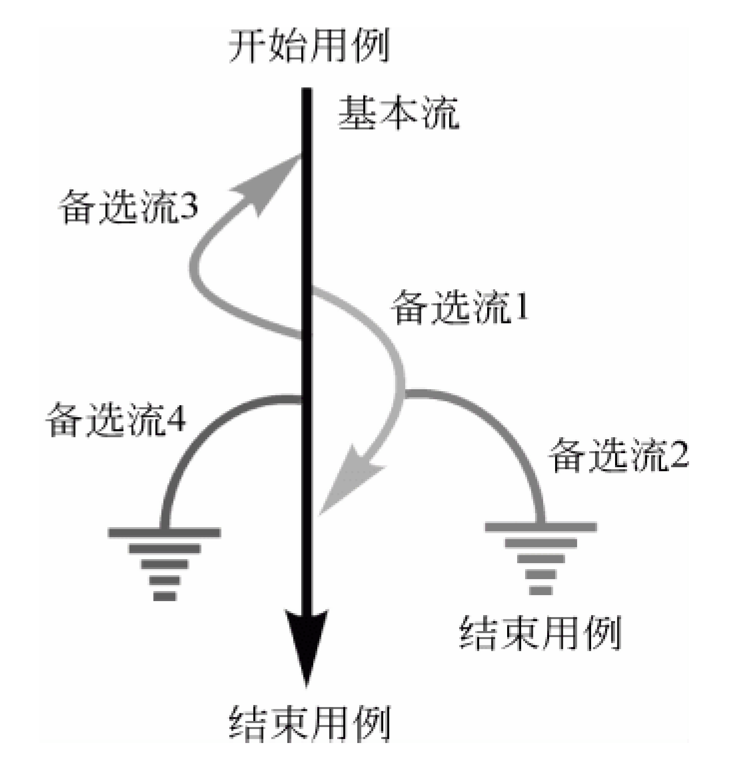


图 4.11 基本流和备选流

场景法的基本设计步骤如下所示：

① 根据说明，描述程序的基本流及各项备选流。

② 根据基本流和各项备选流生成不同的场景。

③ 对每一个场景生成相应的测试用例。

④ 对生成的所有测试用例重新复审，去掉多余的测试用例，测试用例确定后，对每一个测试用例确定测试数据值。

图 4.11 中，有一个基本流和四个备选流。每个经过用例的可能路径，确定不同的用例场景。从基本流开始，再将基本流和备选流结合起来，可以确定以下用例场景：

场景 1：基本流

场景 2：基本流→备选流 1

场景 3：基本流→备选流 1→备选流 2

场景 4：基本流→备选流 3

场景 5：基本流→备选流 3→备选流 1

场景 6：基本流→备选流 3→备选流 1→备选流 2

场景 7：基本流→备选流 4

场景 8：基本流→备选流 3→备选流 4

4.6.2 应用举例

【例 4-9】 采用场景法设计 ATM 系统的测试用例。

【解析】

（1）例子描述

图 4.12 是 ATM 系统的流程示意图。

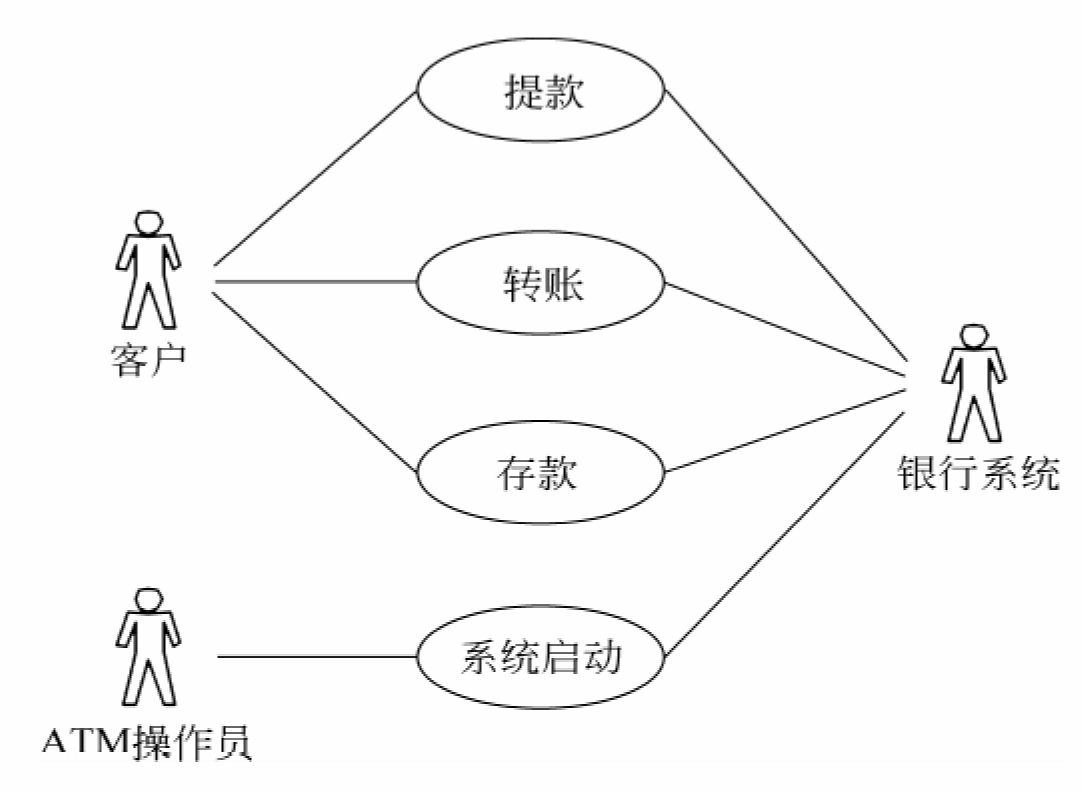


图 4.12 ATM 流程示意图

（2）场景设计

图 4.9 中所示 ATM 系统中的基本流和某些备用流如表 4.11 所示。

表 4.11 基本流和备选流

从 ATM 机上取款的流程		
基本流	步骤 1	准备提款：客户将银行卡插入 ATM 机的读卡机
	步骤 2	验证银行卡：ATM 机从银行卡的磁条中读取账户代码，并检查它是否属于可以接收的银行卡
	步骤 3	输入 PIN 码（4 位）验证账户代码和 PIN，验证账户代码和 PIN，以确定该账户是否有效以及所输入的 PIN 对该账户来说是否正确
	步骤 4	ATM 选项：ATM 显示在本机上可用的各种选项。在此事件流中，银行客户通常选择“提款”
	步骤 5	输入金额：要从 ATM 中提取的金额。对于此事件流，客户需选择预设的金额（10 元、20 元、50 元或 100 元）。授权 ATM 通过将卡 ID、PIN、金额以及账户信息作为一笔交易发送给银行系统来启动验证过程。对于此事件流，银行系统处于联机状态，而且对授权请求给予答复，批准完成提款过程，并且据此更新账户余额
	步骤 6	出钞：提供现金
	步骤 7	返回银行卡：银行卡被返还
	步骤 8	收据：打印收据并提供给客户。ATM 还相应地更新内部记录
	用例结束时 ATM 又回到准备就绪状态	
备选流 1——银行卡无效	在基本流步骤 2 中验证银行卡，如果卡是无效的，则卡被退回，同时会通告相关消息	
备选流 2——ATM 内没有现金	在基本流步骤 4 中 ATM 选项，选项将无法使用。如果 ATM 内没有现金，则“提款”选项不可用	

续表

从 ATM 机上取款的流程	
备选流 3——ATM 内现金不足	在基本流步骤 5 中输入金额，如果 ATM 机内金额少于请求提取的金额，则将显示一则适当的消息，并且在步骤 6 输入金额处重新加入基本流
备选流 4——PIN 有误	在基本流步骤 3 中验证账户和 PIN，客户有三次机会输入 PIN。如果 PIN 输入有误，ATM 将显示适当的消息；如果还存在输入机会，则此事件流在步骤 3 输入 PIN 处重新加入基本流。如果最后一次尝试输入的 PIN 码仍然错误，则该卡将被 ATM 机保留，同时 ATM 返回到准备就绪状态，本用例终止
备选流 5——账户不存在	在基本流步骤 3 中验证账户和 PIN，如果银行系统返回的代码表明找不到该账户或禁止从该账户中提款，则 ATM 显示适当的消息并且在步骤 8 返回银行卡处重新加入基本流
备选流 6——账面金额不足	在基本流步骤 6 授权中，银行系统返回代码表明账户余额少于在基本流步骤 5 输入金额内输入的金额，则 ATM 显示适当的消息并且在步骤 5 输入金额处重新加入基本流
备选流 7——达到每日最大的提款金额	在基本流步骤 6 授权中，银行系统返回的代码表明包括本提款请求在内，客户已经或将超过在 24 小时内允许提取的最多金额，则 ATM 显示适当的消息并在步骤 5 输入金额上重新加入基本流
备选流 x——记录错误	如果在基本流步骤 9 收据中，记录无法更新，则 ATM 进入“安全模式”，在此模式下所有功能都将暂停使用。同时向银行系统发送一条适当的警报信息，表明 ATM 已经暂停工作
备选流 y——退出	客户可随时决定终止交易（退出）。交易终止，银行卡随之退出
备选流 z——“翘起”	ATM 包含大量的传感器，用以监控各种功能，如电源检测器、不同的门和出入口处的测压器以及动作检测器等。在任一时刻，如果某个传感器被激活，则警报信号将发送给警方而且 ATM 进入“安全模式”，在此模式下所有功能都暂停使用，直到采取适当的重启/重新初始化的措施
第一次迭代中，根据迭代计划，我们需要核实提款用例已经正确地实施。此时尚未实施整个用例，只实施了下面的事件流： 基本流——提取预设金额（10 元、20 元、50 元、100 元） 备选流 2——ATM 内没有现金 备选流 3——ATM 内现金不足 备选流 4——PIN 有误 备选流 5——账户不存在/账户类型有误 备选流 6——账面金额不足	

表 4.12 所示是 ATM 生成的场景。

表 4.12　场景设计

场　　景	处 理 流 程
场景 1——成功提款	基本流
场景 2——ATM 内没有现金	基本流——备选流 2
场景 3——ATM 内现金不足	基本流——备选流 3
场景 4——PIN 有误（还有输入机会）	基本流——备选流 4
场景 5——PIN 有误（不再输入机会）	基本流——备选流 4
场景 6——账户不存在/账户类型有误	基本流——备选流 5
场景 7——账户余额不足	基本流——备选流 6

注：为方便起见，备选流 3 和 6（场景 3 和 7）内的循环以及循环组合未纳入表 4.12。

（3）用例设计

对于这 7 个场景中的每一个场景都需要确定测试用例，一般采用矩阵或决策表来确定和管理测试用例。例 4-6 中测试用例包含测试用例 ID、场景/条件、测试用例中涉及的所有数据元素和预期结果等项目。首先确定执行用例场景所需的数据元素，然后构建矩阵，最后要确定包含执行场景所需的适当条件的测试用例。表 4.13 中"行"代表各个测试用例，"列"代表测试用例的信息。V 表示这个条件必须是有效的才可执行基本流，I 表示这种条件下将激活所需备选流 ，n/a 表示这个条件不适用于测试用例。

表 4.13 测试用例表

TC(测试用例)ID 号	场景/条件	PIN	账号	输入(或选择)的金额	账面金额	ATM 内的金额	预期结果
CW1	场景 1：成功提款	V	V	V	V	V	成功提款
CW2	场景 2：ATM 内没有现金	V	V	V	V	I	提款选项不可用，用例结束
CW3	场景 3：ATM 内现金不足	V	V	V	V	I	警告消息，返回基本流步骤 5，输入金额
CW4	场景 4：PIN 有误(还有不止一次输入机会)	I	V	n/a	V	V	警告消息，返回基本流步骤 3，输入 PIN
CW5	场景 4：PIN 有误(还有一次输入机会)	I	V	n/a	V	V	警告消息，返回基本流步骤 3，输入 PIN

表 4.13 中，六个测试用例执行了四个场景。测试用例 CW1 被称为正面测试用例，它一直沿着用例的基本流路径执行。基本流的全面测试必须包括负面测试用例，以确保只有在符合条件的情况下才执行基本流。这些负面测试用例由 CW2～CW6 表示。

每个场景只有一个正面测试用例和负面测试用例是不充分的，场景 4 正是这样的一个示例。要全面地测试场景 4，至少需要三个正面测试用例，以激活场景 4：

① 输入了错误的 PIN，但仍存在输入机会，此备选流重新加入基本流中的步骤 3-输入 PIN。

② 输入了错误的 PIN，而且不再有输入机会，则此备选流将保留银行卡并终止用例。

③ 最后一次输入了"正确"的 PIN。备选流在步骤 5 输入金额处重新加入基本流。

注意：表 4.8 中，无须为条件输入任何实际的值。以这种方式创建测试用例矩阵的一个优点在于容易看到测试的是什么条件。由于只需要查看 V 和 I，这种方式还易于判断是否已经确定了充足的测试用例。

（4）数据设计

一旦确定了所有的测试用例，则应对这些用例进行复审和验证以确保其准确且适度，并取消多余或等效的测试用例，如表 4.14 所示。

测试用例一经认可，就可以确定实际数据值并且设定测试数据。以上测试用例只是在本次迭代中需要用来验证提款用例的一部分测试用例。需要的其他测试用例包括以下内容。

表 4.14　测试用例表

TC(测试用例)ID号	场景/条件	PIN	账号	输入(或选择)的金额(元)	账面金额(元)	ATM内的金额(元)	预期结果
CW1	场景1：成功提款	4987	678-498	50.00	500.00	2000	成功提款。账户余额被更新为450.00
CW2	场景2：ATM内没有现金	4987	678-498	100.00	500.00	0.00	提款选项不可用，用例结束
CW3	场景3：ATM内现金不足	4987	678-498	100.00	500.00	70.00	警告消息，返回基本流步骤5，输入金额
CW4	场景4：PIN有误(还有不止一次输入机会)	4978	678-498	n/a	500.00	2 000	警告消息，返回基本流步骤3，输入PIN
CW5	场景4：PIN有误(还有一次输入机会)	4978	678-498	n/a	500.00	2 000	警告消息，返回基本流步骤3，输入PIN
CW6	场景4：PIN有误(不再有输入机会)	4978	678-498	n/a	500.00	2 000	警告消息，卡予保留，用例结束

场景6——账户不存在/账户类型有误：未找到账户或账户不可用。

场景6——账户不存在/账户类型有误：禁止从该账户中提款。

场景7——账户余额不足：请求的金额超出账面金额。

4.7　综合策略

黑盒测试方法有等价类划分、边界值分析、决策表、因果图、场景法等，每种测试方法都有其各自的特点和适用场合。

等价类划分是通过等价类划分减少测试用例的绝对数量，适用于强数据类型语言编程的“输入—处理—输出”结构化的程序体系结构，但等价类划分只是机械地从对应等价类中选择输入值而不考虑其应用领域的相关知识。例如，NextDate函数含有三个变量(year、month、day)，由于日期、月份和年变量之间存在相互依赖关系，对于2月和闰年的测试，等价类划分方法就不充分。

边界值分析通过分析输入变量的边界值域设计测试用例，适合于当被测程序含有多个独立变量的函数，而且这些变量受物理量的限制的情况。边界值分析对布尔变量和逻辑变量没有多大意义。

在基于决策表的测试中，通过分析被测程序的逻辑依赖关系，构造决策表，进而设计测试用例。

等价类划分、边界值分析和决策表方法生成测试用例的数量与开发测试用例所需工作量的对比如图4.13所示。

软件测试专家Myers给出了黑盒测试方法中各种测试方法的使用策略：

(1) 在任何情况下都必须使用边界值分析方法。经验表明，用这种方法设计的测试

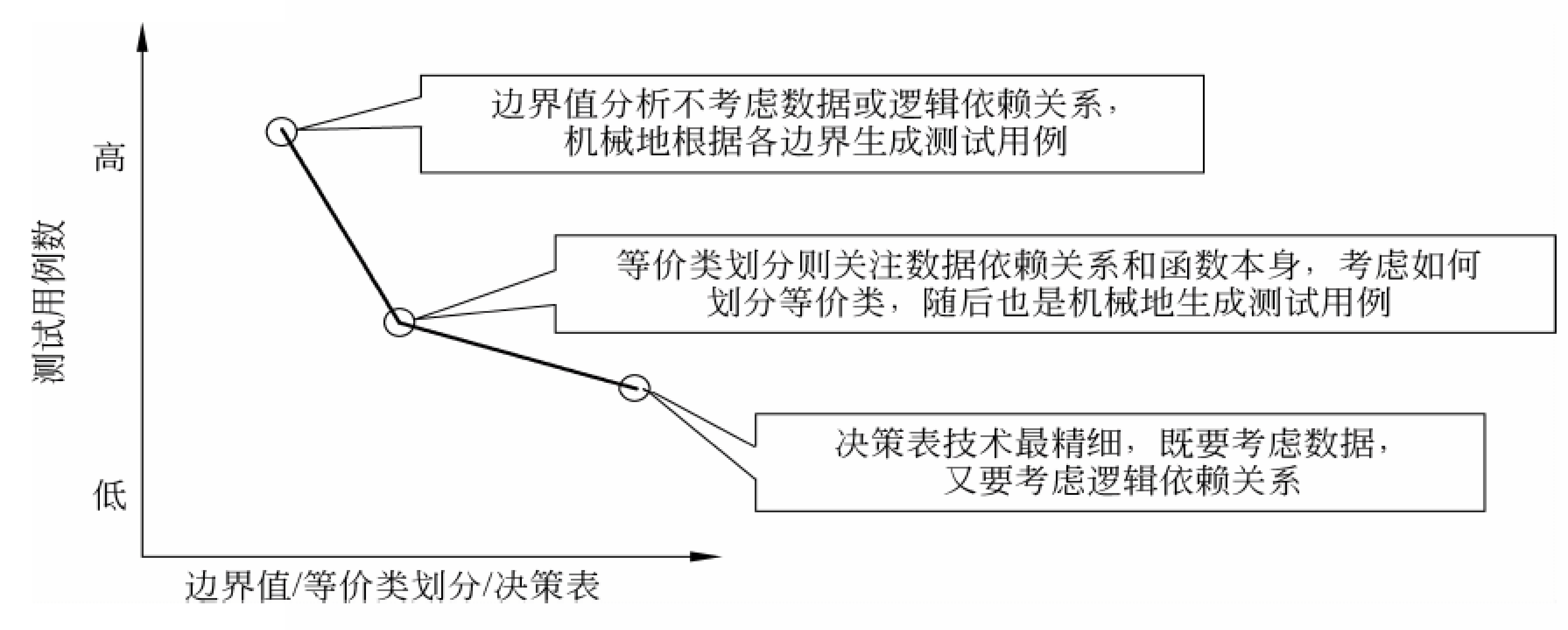


图 4.13 测试用例的数量与开发测试用例所需工作量的对比图

用例发现程序错误的能力最强。

（2）必要时使用等价类划分方法补充一些测试用例。

（3）用错误逻辑法追加一些测试用例。

（4）对照程序逻辑，检查已设计出的测试用例的逻辑覆盖程度，如果没有达到要求的覆盖标准，应当再补充足够的测试用例。

（5）如果程序的功能说明中含有输入条件的组合情况，则一开始就可选用因果图法。

总之，对于功能性测试技术可以根据如下条件进行选择：

（1）如果变量引用的是物理量，可采用定义域测试和等价类测试。

（2）如果变量是独立的，则可以用定义域测试和等价类测试。

（3）如果变量不是独立的，可采用决策表测试。

（4）如果为单缺陷假设，则可采用边界值分析和健壮性测试。

（5）如果为多缺陷假设，可采用最坏情况测试、健壮最坏情况测试和决策表测试。

（6）如果程序包含大量例外处理，可采用健壮性测试和决策表测试。

（7）如果变量引用的是逻辑量，可采用等价类测试用例和决策表测试。

4.8 习　　题

1. 选择题

（1）黑盒测试是通过软件的外部表现来发现软件缺陷和错误的测试方法，具体地说，黑盒测试用例设计技术包括________等。

A. 等价类划分法、因果图法、边界值分析法、错误推测法、判定表驱动法

B. 等价类划分法、因果图法、边界值分析法、正交试验法、符号法

C. 等价类划分法、因果图法、边界值分析法、功能图法、基本路径法

D. 等价类划分法、因果图法、边界值分析法、静态质量度量法、场景法

（2）常用的黑盒测试方法有边值分析、等价类划分、错误猜测、因果图等。其中________经常与其他方法结合起来使用。

A. 边值分析　　B. 等价类划分　　C. 错误猜测　　D. 因果图

(3) 等价类划分完成后,就可得出________,它是确定测试用例的基础。
A. 有效等价类　B. 无效等价　C. 等价类表　D. 测试用例集
(4) 在设计测试用例时,________是用得最多的一种黑盒测试方法。
A. 等价类划分　B. 边界值分析　C. 因果图　D. 功能图
(5) 在黑盒测试中,着重检查输入条件的组合的测试用例设计方法是________。
A. 等价类划分　B. 边界值分析　C. 错误推测法　D. 因果图法
(6) 除了测试程序外,黑盒测试还适用于对________阶段的软件文档进行测试。
A. 编码　B. 软件详细设计　C. 软件总体设计　D. 需求分析
(7) 由因果图转换出来的________是确定测试用例的基础。
A. 判定表　B. 约束条件表　C. 输入状态表　D. 输出状态表

2. 判断题

(1) 用黑盒法测试时,测试用例是根据程序内部逻辑设计的。( )
(2) 黑盒测试方法中最有效的是因果图法。( )
(3) 对于连锁型分支结构,若有 n 个判定语句,则有 $2n$ 条路径。( )
(4) 尽量采用复合的条件测试,以避免嵌套的分支结构。( )
(5) GOTO 语句概念简单,使用方便,在某些情况下,保留 GOTO 语句反能使写出的程序更加简洁。( )

3. 简答题

(1) 等价类划分的原则是什么?
(2) 边界值分析的设计原则是什么? 有函数 $f(x,y,z)$,其中,$x\in[1900,2100]$,$y\in[1,12]$,$z\in[1,31]$。请写出该函数采用边界值分析法设计的测试用例。
(3) 采用等价类划分方法设计三角形类型的测试用例。
(4) 设计一个日期函数的决策表测试用例。



白盒测试

本章介绍了白盒测试的相关内容，包括白盒测试的历程、逻辑覆盖法、路径分析、数据流测试和程序插桩等。其中，逻辑覆盖法包括语句覆盖、判定覆盖、条件覆盖、条件判定覆盖、修正条件判定覆盖和条件组合覆盖。路径分析包括基路径测试和循环测试。数据流测试包括变量定义/引用分析和程序片方法。

5.1 概　　述

白盒测试是把测试对象看做一个打开的盒子，允许测试人员利用程序内部的逻辑结构及有关信息，设计或选择测试用例，通过在不同点检查程序状态，确定实际状态是否与预期的状态一致。白盒测试软件产品的内部结构和处理过程，而不测试软件产品的功能，用于纠正软件系统在描述、表示和规格上的错误。

白盒测试大致经历了四代变迁。测试发展初期，人们通常以单步调试代替测试，或采用 assert 断言、print 语句等简单方式的进行测试。这一时期的测试是半手工的，没实现自动化，测试效果也严重依赖测试者的个人能力，缺少统一规范的评判标准，测试过程难以重用，测试结果难以评估与改进。

第 2 代白盒测试将测试操作用形式化语言（也称测试脚本）来表述，脚本可以组合成测试用例，测试用例组合成测试集，测试集用测试工程管理。另外，代码覆盖率功能使测试结果可以进行评估，直观地看到哪些代码或分支未被覆盖，从而进行针对性的测试。目前市面上 CodeTest、Visual Tester、C++ Tester 等都属于第 2 代白盒测试工具。

第 3 代白盒测试解决了重复测试问题，使得测试操作被规范格式记录，当被测对象没变化，或变化很少时，测试用例可以反复重用。当然，如果源码大幅调整，甚至重构，如何维持测试用例同步更新，则第 3 代白盒测试技术仍无法解决。第 3 代白盒测试工具以 xUnit 为代表，包括 JUnit、DUnit、CppUnit 等。

第 4 代白盒测试方法相对第 3 代白盒测试方法而言，将测试设计、执行与改进等测试过程融入到软件的整个开发全过程，解决了持续测试的问题。

从评估测试效果、自动测试、持续测试和调测一体等几个方面分析白盒测试，可得表 5.1。

表 5.1　白盒测试的历程

	评估测试效果	自动测试	持续测试	调测一体
第 1 代	否	否	否	否
第 2 代	是	是	否	否
第 3 代	是	是	是	否
第 4 代	是	是	是	是

当前，常用的白盒测试方法有代码检查法、静态结构分析法、静态质量度量法、逻辑覆盖法、基本路径测试法、数据流测试、域测试、符号测试、Z 路径覆盖、程序插桩。下面介绍其中几个流行的测试方法。

5.2　逻辑覆盖法

逻辑测试方法，又称为控制流覆盖，是一种按照程序内部逻辑结构和编码结构设计测试用例的测试方法，目的是要测试程序中的语句、判定(控制流能够分解为不同路径的程序点)、条件(形成判定的原子谓词)等。根据覆盖的标准不同，分为语句覆盖、判定覆盖、条件覆盖、条件判定覆盖、修正条件判定覆盖、增强条件判定覆盖、条件组合覆盖和路径覆盖等。

下面，通过例 5-1 讲解逻辑覆盖的各种测试方法。

【例 5-1】　用 C++ 实现简单的数学运算。

【解析】　代码如下所示：

```
Dim a,b As Integer
  Dim c As Double
  If (a>0 And b>0) Then
    c=c/a
  End if
  If (a>1 or c>1) Then
    c=c+1
End if
c=b+c
```

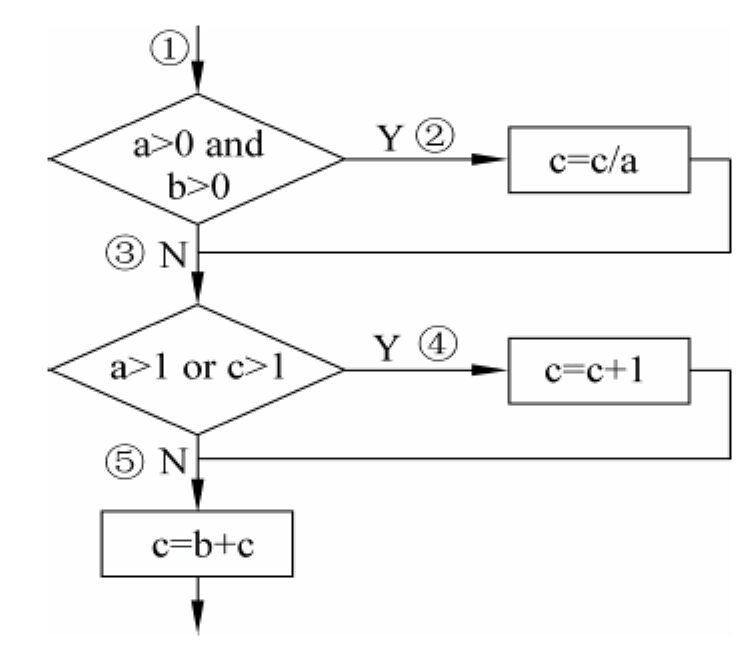


图 5.1　程序流程图

例 5-1 的流程图如图 5.1 所示。其中①,②,…,⑤是控制流上若干程序点。

5.2.1　语句覆盖

语句覆盖又称为线覆盖面或段覆盖面。其含义是指，选择足够数目测试数据，使被测程序中每条语句至少执行一次。

语句覆盖设计例 5-1 的测试用例 a=2,b=2,c=4，则程序按照路径①—>②—>③—>④—>⑤执行，程序段中的 5 个语句均执行，符合语句覆盖。但是，如果测试用例选择 a=2,b=—2,c=4，程序按照路径①—>③—>④—>⑤执行，则未能达到语句覆盖。

语句覆盖测试方法仅仅针对程序逻辑中显式语句，对隐藏条件无法测试。若将例 5-1 中第一个逻辑运算符 and 误写成 or，测试用例 a=2，b=2，c=4 仍能达到语句覆盖的要求，但是并未发现程序中误写错误。

语句覆盖可以直接应用于目标代码，不需要处理源代码。作为最弱逻辑覆盖，语句覆盖对一些控制结构不敏感，不能发现判断中逻辑运算符的错误，覆盖率很低。

5.2.2 判定覆盖

判定覆盖又称为分支覆盖或所有边覆盖，测试控制结构中布尔表达式分别为真和假（例如 if 语句和 while 语句）。布尔型表达式被认为一个整体，取值为 True 或 False，而不考虑内部是否包含“逻辑与”或者“逻辑或”等操作符。

判定覆盖的基本思想是指设计的测试用例，使得程序中每个判定至少分别取“真”分支和取“假”分支至少经历一次，即判断真假值均被满足。

判定覆盖设计例 5-1 的测试用例如表 5.2(a)或表 5.2(b)所示。

表 5.2(a) 例 5-1 的判定覆盖测试用例

测试用例	a>0 and b>0	a>1 or c>1	执行路径
a=1，b=1，c=5	T	T	Ⅰ→Ⅱ→Ⅲ→Ⅳ→Ⅴ
a=1，b=−2，c=−3	F	F	Ⅰ→Ⅲ→Ⅴ

表 5.2(b) 例 5-1 的判定覆盖测试用例

测试用例	a>0 and b>0	a>1 or c>1	执行路径
a=1，b=1，c=−3	T	F	Ⅰ→Ⅱ→Ⅲ→Ⅴ
a=1，b=−2，c=3	F	T	Ⅰ→Ⅲ→Ⅳ→Ⅴ

判定覆盖作为语句覆盖的超集，比语句覆盖要多几乎一倍的测试路径，当然也就具有比语句覆盖更强的测试能力。由于大部分的判定语句是由多个逻辑条件组合而成（如判定语句中包含 AND、OR、CASE），判定覆盖仅仅判断其整个最终结果，而无法发现判定内部的每个条件的取值情况，因此，必然会遗漏部分测试路径。

分析下面一段 C 语言代码。

```
If (condition1 && (condition2 || function1()))
    Statement1;
Else
    Statement2;
```

当判定 condition1 和 condition2 取值为真时，执行 Statement1 表达式；当判定 condition1 和 condition2 取值为假，则执行 Statement2 表达式，可知，只需判定 condition1 取值为假，condition2 取值不管为何，执行 Statement2 表达式，在这段代码的控制结构中，操作符“||”排除 function1 函数的影响，不用考虑函数 function1 的调用。因此，当判定语句由多个逻辑条件组合而成，判定覆盖仅仅判断最终结果，而忽略每个条件的取值情况，从而必然会遗漏部分测试路径。

5.2.3 条件覆盖

条件覆盖是设计测试用例，使每个判断中每个条件的可能取值至少满足一次。

条件覆盖设计例 5-1 的测试用例，针对 a>0 and b>0 判定条件表达式，a>0 取值为“真”，记为 T1；a>0 取值为“假”，记为 F1；b>0 取值“真”，记为 T2 ；b>0 取值为“假”，记为 F2；条件表达式 a>1 or c>1，a>1 取值为“真”，记为 T3 ；a>1 取值为“假”，记为 F3；c>1 取值为“真”，记为 T4；c>1 取值为“假”，记为 F4。如表 5.3 所示。

表 5.3 例 5-1 的条件覆盖测试用例

测试用例	覆盖条件	具体取值条件	执行路径
a=2，b=−1，c=−2	T1，F2，T3，F4	a>0，b<=0，a>1，c<=1	①→③→④→⑤
a=−1，b=2，c=3	F1，T2，F3，T4	a<=0，b>0，a<=1，c>1	①→③→④→⑤

条件覆盖只能保证每个条件有一次为真，有一次为假，而不考虑所有的判定结果。表 5.3中的测试用例a=2，b=−1 和测试用例 a=−1，b=2 满足了条件覆盖的测试用例，保证了 a>0 and b>0 两个条件的可能值(True 和 False)至少满足一次，但是，由于测试用例的所有判定结果都是 False，并没有满足判定覆盖。故，条件覆盖不一定包含判定覆盖。

5.2.4 条件判定覆盖

既然判定条件不一定包含条件覆盖，条件覆盖也不一定包含判定覆盖，自然会提出一种能同时满足两种覆盖标准的逻辑覆盖，这就是条件判定覆盖。(Condition/Decision Coverage，C/DC)。其英文原文：Condition/Decision Coverage—it combines the requirements for decision coverage with chose for condition coverage. That is，there must be sufficient test cases to toggle the decision outcome between true and false and to toggle each condition value between true and false。解释为：条件判定覆盖的含义是通过设计足够的测试用例，使得判断条件中的所有条件可能至少执行一次取值，同时，所有判断的可能结果至少执行一次。因此，条件判定覆盖的测试用例满足如下条件：

(1) 所有条件可能至少执行一次取值。

(2) 所有判断的可能结果至少执行一次。

例 5-1 的条件判定覆盖测试用例如表 5.4 所示。

表 5.4 例 5-1 的条件判定覆盖测试用例

测 试 用 例	覆盖条件	执 行 路 径
a=2，b=1，c=5	T1，T2，T3，T4	①→②→③→④→⑤
a=−1，b=−2，c=−3	F1，F2，F3，F4	①→③→⑤

条件判定覆盖能同时满足判定、条件两种覆盖标准，是判定和条件覆盖设计方法的交集，具有两者的简单性却没有两者的缺点。表面上，条件判定覆盖测试了所有条件的取

值,但事实并非如此,往往某些条件掩盖了另一些条件,并没有覆盖所有的 true 和 false 取值的条件组合情况,会遗漏某些条件取值错误的情况。为彻底地检查所有条件的取值,需要将判定语句中给出的复合条件表达式进行分解,形成由多个基本判定嵌套的流程图,方可有效地检查所有的条件是否正确。

5.2.5 修正条件判定覆盖

修正条件判定覆盖(Modified Condition/Decision Coverage,MC/DC)的英文原文:Modified Condition/Decision Coverage—every point of entry and exit in the program has been invoked at least once, every condition in the program has taken all possible outcomes at least once,and each condition in a decision has been shown to independently affect a decision′s outcome by varying just that condition while holding fixed all other possible conditions。解释为:更改的判定/条件覆盖是判定中每个条件的所有可能结果至少出现一次,每个判定本身的所有可能结果也至少出现一次,每个入口点和出口点至少要唤醒一次,并且每个条件都显示能单独影响判定结果。

MC/DC 定义的第一和第二部分是条件/判定覆盖准则,其后是 MC/DC 特有的判定条件。定义中最关键的字是"独立影响",也就说明每一次每一个判定条件发生改变,必然会导致一次判定结果的改变,消除判定中的某些条件会被其他的条件所掩盖的问题,从而使得测试更加完备。MC/DC 的目的就是消除测试过程中的各个单独条件之间的相互影响并且保证每个单独条件能够分别影响判定结果的正确性。

例如,A OR B 全部测试用例组合如表 5.5 所示。

表 5.5 A OR B 全部测试用例组合表

测试用例	A	B	结果
1	T	T	T
2	T	F	T
3	F	T	T
4	F	F	F

注:为描述方便,T 表示条件为真(True). F 表示条件为假(False)。

测试用例对(2,4)说明条件 A 独立地影响测试结果,测试用例对(3,4)说明条件 B 独立地影响测试结果,所以采用测试用例对(2, 3,4)进行测试,满足 MC/DC 覆盖准则。

MC/DC 继承了语句覆盖准则,条件/判定覆盖准则,多重条件覆盖等判定条件,同时加入了新的判定条件。例如,表 5.5 中 A OR B 误写为 A AND B,因为 $T\cap T=T\cup T$,且 $F\cap F=F\cup F$,两者所得到的判定结果相同,由此可说明虽然使用了判定条件覆盖(C/DC)准则来测试语句,此错误仍然不能检测出来。但如何使用 MC/DC 方法,就可以发现这样的错误,原因是:$T\cup F$ 值为 T,而 $T\cap F$ 值为 F,由此可说明中间的操作符号发生了错误。MC/DC 具有如下优点:

(1) 继承了多重条件覆盖的优点。

(2) 线性地增加了测试用例的数量。

（3）对操作数及非等式条件变化反应敏感。

（4）具有更高的目标码覆盖率。

在许多软件系统中，尤其是以嵌入式和实时性为特征的航空机载软件中 MC/DC 得到广泛的应用，如：MC/DC 已经被应用于 RTCA/DO-178B 标准当中，这个标准主用于美国测试飞行软件的安全性的审查。

5.2.6 条件组合覆盖

条件组合覆盖的基本思想：设计测试用例使得判断中每个条件的所有可能至少出现一次，并且每个判断本身的判定结果也至少出现一次，与条件覆盖的差别是条件组合覆盖不是简单地要求每个条件都出现“真”与“假”两种结果，而是要求这些结果的所有可能组合都至少出现一次。

条件组合覆盖是一种相当强的覆盖准则，可以有效地检查各种条件取值的组合情况是否正确。条件组合覆盖不但可覆盖所有条件的可能取值的组合，还可覆盖所有判断的可取分支。

例 5-1 的条件组合覆盖测试用例如表 5.6 和表 5.7 所示。

表 5.6 例 5-1 的条件组合覆盖测试用例

编号	覆盖条件取值	判定条件取值	具体条件取值
1	T1,T2	表达式 a>0 And b>0 取 Y	a>0,b>0
2	T1,F2	表达式 a>0 And b>0 取 N	a>0,b<=0
3	F1,T2	表达式 a>0 And b>0 取 N	a<=0,b>0
4	F1,F2	表达式 a>0 And b>0 取 N	a<= 0,b<=0
5	T3,T4	表达式 a>1 or c>1 取 Y	a>1,c>1
6	T3,F4	表达式 a>1 or c>1 取 Y	a>1,c<=1
7	F3,T4	表达式 a>1 or c>1 取 Y	a<=1,c>1
8	F3,F4	表达式 a>1 or c>1 取 N	a<=1,c<=1

表 5.7 例 5-1 的条件组合覆盖测试用例

测试用例	覆盖条件	覆盖判断	覆盖组合	执行路径
a=2,b=1,c=5	T1,T2, T3,T4	表达式 a>0 And b>0 取 Y 分支 表达式 a>1 or c>1 取 Y 分支	编号 1,编号 5	①→②→③→ ④→⑤
a=2,b=−1, c=−2	T1,F2, T3,F4	表达式 a>0 And b>0 取 N 分支 表达式 a>1 or c>1 取 Y 分支	编号 2,编号 5	①→③→④→⑤
a=−1,b=2, c=3	F1,T2, F3,T4	表达式 a>0 And b>0 取 N 分支 表达式 a>1 or c>1 取 Y 分支	编号 3,编号 7	①→③→④→⑤
a=−1,b=−2, c=−3	F1,F2, F3,F4	表达式 a>0 And b>0 取 N 分支 表达式 a>1 or c>1 取 N 分支	编号 4,编号 8	①→③→⑤

条件组合覆盖准则满足判定覆盖、条件覆盖和判定/条件覆盖准则，线性地增加了测试用例的数量，却不能保证所有的路径被执行测试，仍有可能有部分路径被遗漏。如表 5.7 所示，覆盖条件虽然不同，但出现了①→③→④→⑤相同的执行路径，缺少了①→②→③→⑤路径。

5.2.7 路径覆盖

路径覆盖的基本思想：选择足够的测试用例，使得程序中所有的可能路径都至少被执行一次。

例 5-1 的路径覆盖测试用例如表 5.8 所示。

表 5.8 例 5-1 的路径覆盖测试用例

测试用例	覆盖组合	执行路径
a=2,b=1,c=5	编号 1,编号 5	①→②→③→④→⑤
a=1,b=1,c=−3	编号 1,编号 8	①→②→③→⑤
a=−1,b=2,c=3	编号 3,编号 7	①→③→④→⑤
a=−1,b=−2,c=−3	编号 4,编号 8	①→③→⑤

路径覆盖比前面几种逻辑覆盖方法覆盖率都大，但随着程序代码复杂度的增加，测试工作量将呈指数级增长。例如：包含 10 个 if 语句的代码，就有 $2^{10}=1024$ 个路径要测试，如果再增加一个 if 语句，就有 $2^{11}=2048$。

5.2.8 综合举例

【例 5-2】 使用逻辑覆盖法测试如下程序段。

```
void work(int x,int y,int z) {
  int k=0,j=0;
  if((x>3) && (z<10)){
    k=x*y-1;
    j=k-z;
  }
  if((x==4) || (y>5)){
    J=x*y+10;
  }
  j=j%3;
 }
```

【解答】 分析步骤如下：

首先将源代码转化为程序流程图(见图 5.2)和控制流程图(见图 5.3)。

下面，按照不同的覆盖准则设计测试用例。

(1) 语句覆盖

语句覆盖是指设计足够的测试用例使得程序中的每一条可执行语句至少执行一次。故只需保证两个 IF 语句为真，就可以达到语句覆盖的要求。

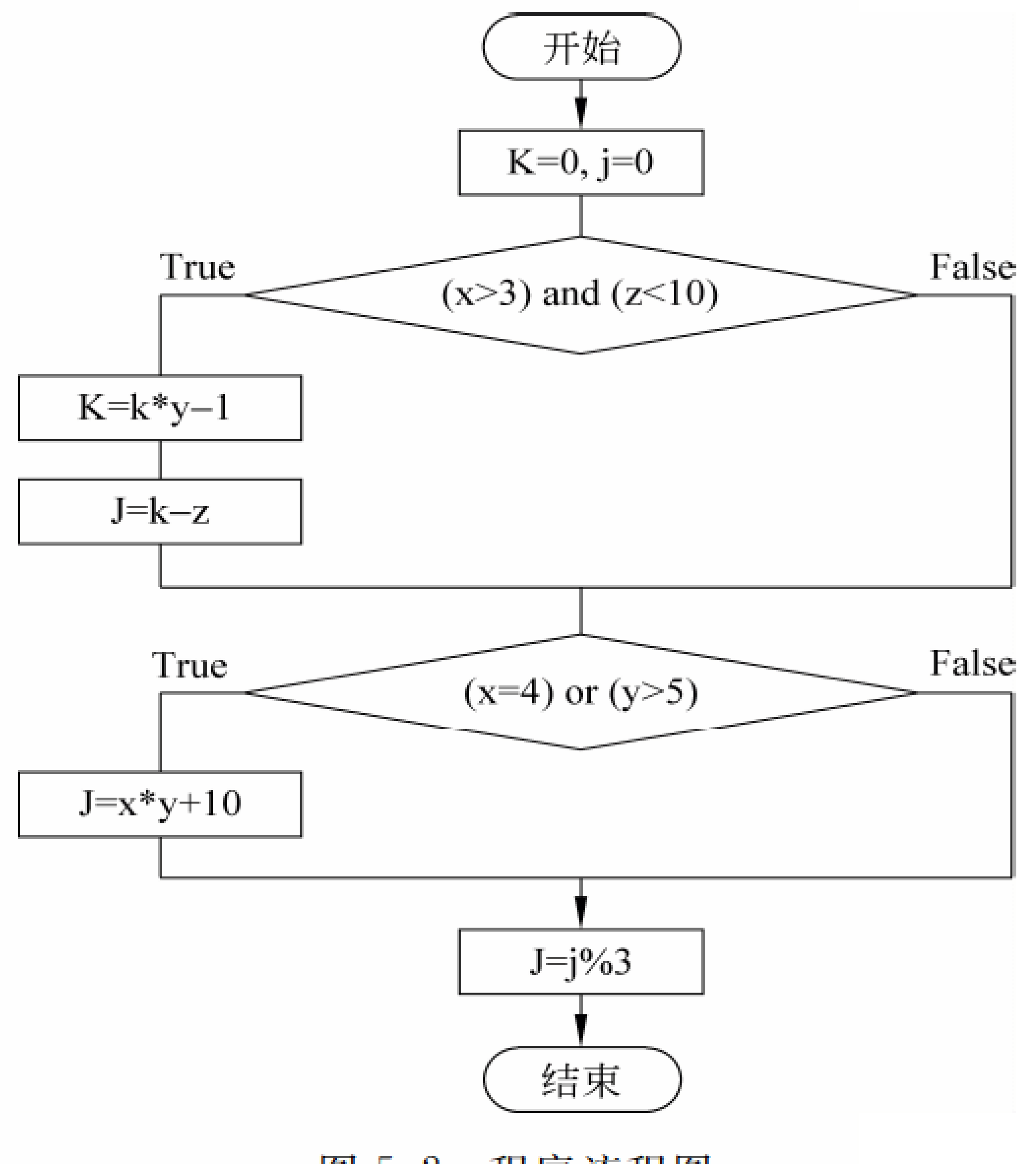


图 5.2　程序流程图

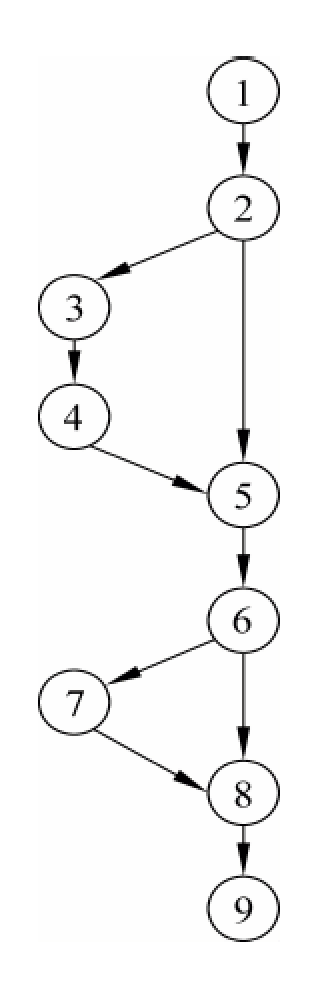


图 5.3　控制流图

设计测试用例如下所示：

x=4、y=5、z=5

【分析】 若在第一个判定((x>3) && (z <10))中把"&&" 错误写成了"||"，而测试用例 x=4、y=5、z=5 仍会按①→②→③→④→⑤→⑥→⑦→⑧路径执行。

(2) 判定覆盖

判定覆盖要求程序中每个判定的取真分支和取假分支至少经历一次。故至少需要设计两组测试用例，分别覆盖两个 IF 语句的真分支和假分支。

判定覆盖设计测试用例如下所示：

- x=4、y=5、z=5，其执行路径为：①→②→③→④→⑤→⑥→⑦→⑧
- x=2、y=5、z=5，其执行路径为：①→②→⑤→⑥→⑧

【分析】 上述的两个测试用例不仅满足了判定覆盖，同时还做到了语句覆盖，故判定覆盖是比语句覆盖更强一些，但仍无法确定判定内部条件的错误。例如，第二个判定中条件 y>5 错误写成了 y<5，而上述的测试用例却仍按照原路径执行将不能发现此处错误。

(3) 条件覆盖

条件覆盖要求每个判断中每个条件的可能取值至少满足一次。本例中，条件有四个：x>3、z<10、x=4 、y>5 ，故设计测试用例使得每个条件分别取真和取假。

① 对于第一个判定 ((x>3) && (z <10))：

- 条件 x>3 取真值记为 T1，取假值记为－T1。
- 条件 z<10 取真值记为 T2，取假值记为－T2。

② 对于第一个判定((x==4)||(y>5)):

- 条件 x==4 取真值记为 T3,取假值记为 -T3。
- 条件 y>5 取真值记为 T4,取假值记为 -T4。

根据条件覆盖的思想,要使上述 4 个条件可能产生的 8 种情况至少满足一次,设计测试用例如表 5.9 所示。

表 5.9 条件覆盖测试用例

测试用例	执行路径	覆盖条件
x=4、y=6、z=5	①->②->③->④->⑤->⑥->⑦->⑧	T1、T2、T3、T4
x=2、y=5、z=15	①->②->⑤->⑥->⑧	-T1、-T2、-T3、-T4

【分析】 表 5.9 中 x=4、y=6、z=5 和 x=2、y=5、z=15 达到了条件覆盖。

若设计如表 5.10 所示的测试用例,虽然也满足了条件覆盖,但只是覆盖了程序中第一个判定的取假分支和第一个判定的取真分支,不满足判定覆盖。

表 5.10 条件覆盖测试用例

测试用例	执行路径	覆盖条件
x=2、y=6、z=5	①->②->⑤->⑥->⑦->⑧	-T1、T2、-T3、T4
x=4、y=5、z=15	①->②->⑤->⑥->⑦->⑧	T1、-T2、T3、-T4

(4) 判定—条件覆盖

判定—条件覆盖要求判断中每个条件的所有可能至少出现一次,并且每个判断本身的可能判定结果也至少出现一次。

判定—条件覆盖设计测试用例如下所示:

① x=4、y=6、z=5,其执行路径为:①→②→③→④→⑤→⑥→⑦→⑧。

② x=2、y=5、z=15,其执行路径为:①→②→⑤→⑥→⑧。

【分析】 从表面上看,判定一条件覆盖测试了各个判定中的所有条件的取值,但实际上,编译器在检查含有多个条件的逻辑表达式时,某些情况下的某些条件会被其他条件所掩盖。例如,对于第一个判定((x>3)&&(z<10)),必须是 x>3 和 z<10 这两个条件同时满足才能确定判定为真,若 x>3 为假,编译器将不再检查 z<10 这个条件,那么若 z<10这个条件书写错误也无法被发现。同样,对((x==4)||(y>5))来说,若条件 x=4 为真,则编译器将判断此判定为真,不再检查 y>5 这个条件了,那么 y>5 这个条件若书写错误也将无法被发现。故判定—条件覆盖也不一定能够完全检查出逻辑表达式的错误。

(5) 条件组合覆盖

条件组合覆盖要求每个判定的所有可能条件取值组合至少执行一次。即,各个判定的条件取值组合标记如下:

① x>3,z<10 记为 T1,T2,第一判定取值为真。

② x>3,z>=10 记为 T1,-T2,第一判定取值为假。
③ x<=3,z<10 记为 -T1,T2,第一判定取值为假。
④ x<=3,z>=10 记为 -T1,-T2,第一判定取值为假。
⑤ x==4,y>5 记为 T3,T4,第二判定取值为真。
⑥ x==4,y<=5 记为 T3,-T4,第二判定取值为真。
⑦ x!=4,y>5 记为 -T3,T4,第二判定取值为真。
⑧ x!=4,y<=5 记为 T3,T4,第二判定取值为假。

根据条件组织覆盖的基本思想,设计测试用例如表 5.11 所示。

表 5.11 条件组合覆盖测试用例

测试用例	执行路径	覆盖条件
x=4、y=6、z=5	①→②→③→④→⑤→⑥→⑦→⑧	T1、T2、T3、T4
x=4、y=5、z=15	①→②→⑤→⑥→⑦→⑧	T1、-T2、T3、-T4
x=2、y=6、z=5	①→②→⑤→⑥→⑦→⑧	-T1、T2、-T3、T4
x=2、y=5、z=15	①→②→⑤→⑥→⑧	-T1、-T2、-T3、-T4

【分析】 表 5.11 中这组测试用例覆盖了所有 8 种条件的组合,覆盖了所有判定的真假分支,但是却丢失了路径①→②→③→④→⑤→⑥→⑧。

(6) 路径覆盖

路径覆盖要求覆盖程序中所有可能的路径。本例中可能的执行路径有四条,故需要四个测试用例。

根据路径覆盖的基本思想,设计测试用例如表 5.12 所示。

表 5.12 路径覆盖测试用例

测试用例	执行路径	覆盖条件
x=4、y=6、z=5	①→②→③→④→⑤→⑥→⑦→⑧	T1、T2、T3、T4
x=4、y=5、z=15	①→②→⑤→⑥→⑦→⑧	T1、-T2、T3、-T4
x=5、y=5、z=5	①→②→③→④→⑤→⑥→⑧	T1、T2、-T3、-T4
x=2、y=5、z=15	①→②→⑤→⑥→⑧	-T1、-T2、-T3、-T4

【分析】 这组测试用例满足了路径覆盖,但并没有覆盖程序中所有的条件组合,丢失了条件组合中的③和⑦,满足路径覆盖的测试用例未必满足条件组合覆盖。

5.3 路径分析

5.3.1 控制流图

程序流程图用于描述程序的结构性,采用不同图形符号标明条件或者处理的有向图,

为了突出控制流结构，将其简化为控制流图，控制流图只有两种图形符号，如图 5.4 所示。

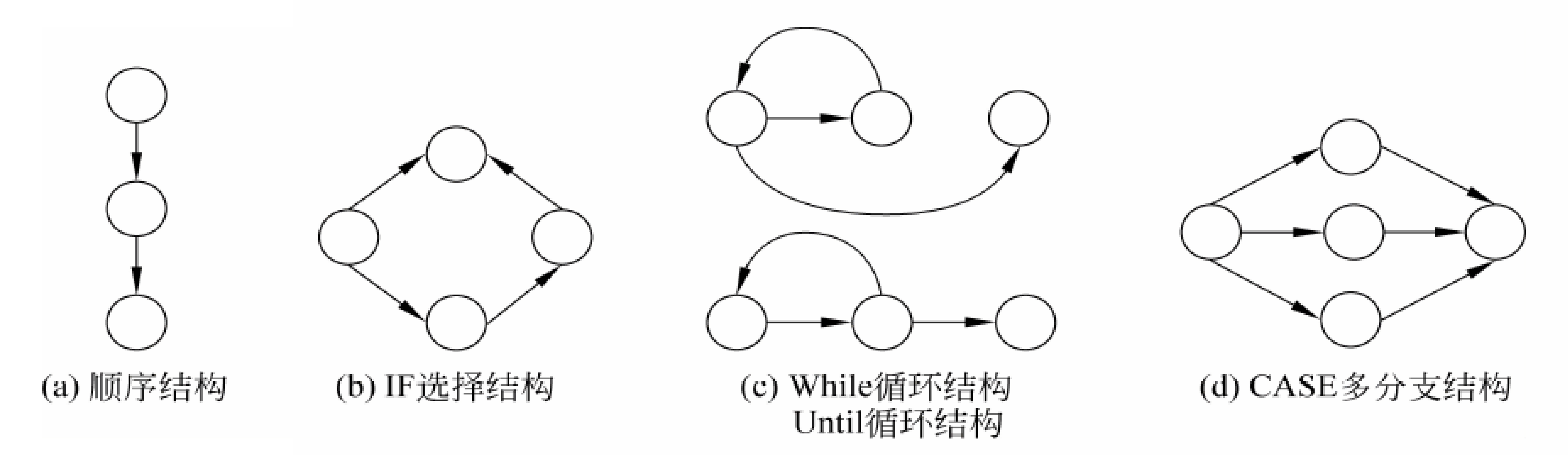


图 5.4　控制流图的基本符号示意图

(1) 结点：以标有编号的圆圈表示，用于表示程序流程图中矩形框、菱形框功能，是一个或多个无分支的语句或源程序语句。

(2) 控制流线或弧：以箭头表示，与程序流程图中的流线功能一致，表示控制的顺序。控制流线通常标有名字，如图中所标的 a、b、c 等。

程序流程图简化成控制流图的过程中，需注意以下情况：

(1) 在选择或多分支结构中，分支的汇聚处应有一个汇聚结点。

(2) 边和结点圈定的区域称为区域。需要注意，图形外的区域也是一个区域。

程序流程图简化成控制流图的过程如图 5.5 所示。

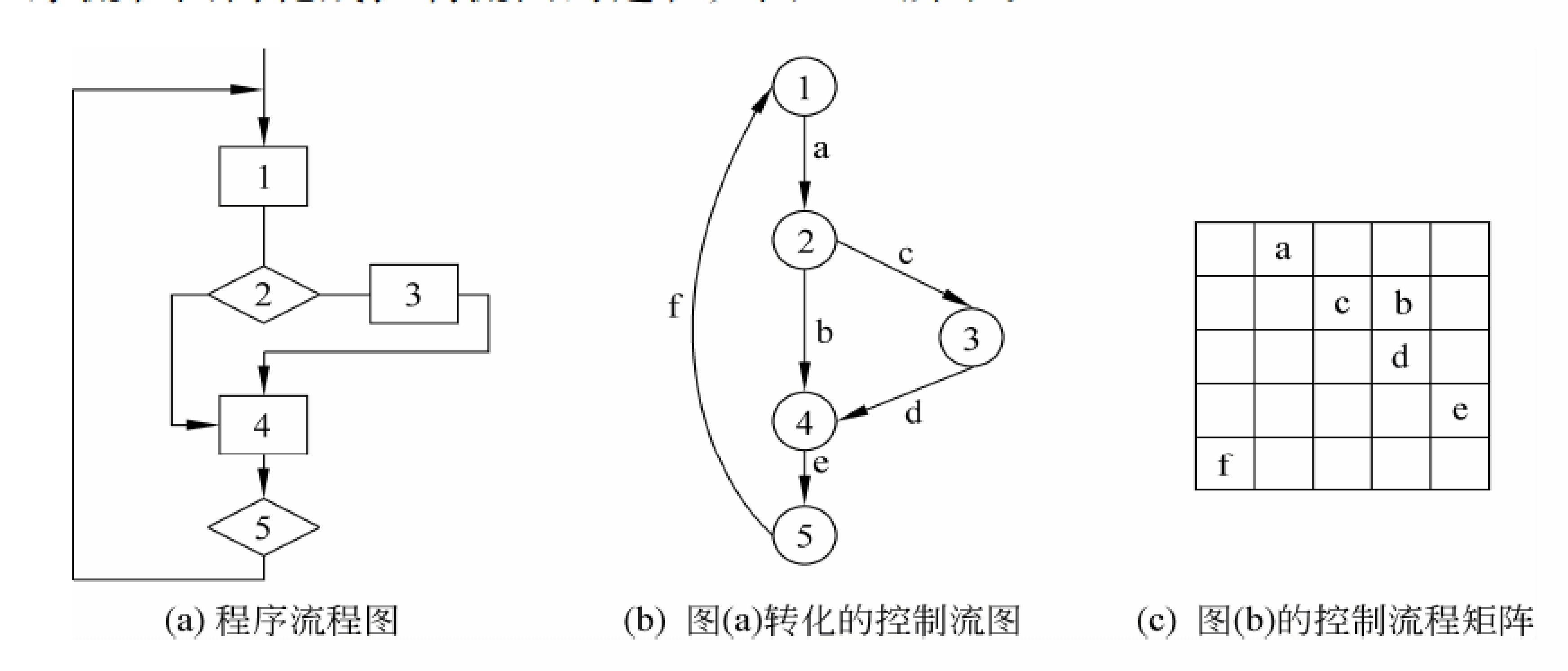


图 5.5　程序流程图转化为控制流图示意图

将控制流图表示成矩阵的形式，称为控制流图矩阵。一个图形矩阵是一个方阵，其行列数目为控制流图中的结点数，行列依次对应到一个被标识的结点，矩阵元素对应到结点间的连接。控制流图的结点用数字标识，边用字母标识，第 i 结点到第 j 结点有 x 边相连接，则对应的图形矩阵中第 i 行与第 j 列有一个非空的元素 x。

图 5.5(c)表示图 5.5(b)的控制流图矩阵。控制流图的 5 个节点决定图 5.5(c)矩阵共有 5 行 5 列。矩阵中 5 个元素 a、b、c、d、e 和 f 的位置对应所在控制流图中的号码。其中，弧 d 在图 5.5(b)中连接了节点 3 至节点 4，故图 5.5(c)的矩阵中处于第 3 行第 4 列。需要注意，控制流图的连接方向，图 5.5(b)中节点 4 到节点 3 没有弧，矩阵中第 4 行第 3 列没有元素。

为了评估程序的控制结构，控制流图矩阵项加入连接权值，连接权值为控制流提供了如下附加信息：

（1）执行连接（边）的概率。

（2）穿越连接的处理时间。

（3）穿越连接时所需的内存。

（4）穿越连接时所需的资源。

最简单情况是连接权值为1（存在连接）或0（不存在连接），如图5.6所示，将控制流图矩阵转化为连接矩阵。字母替换为1，表示存在边。

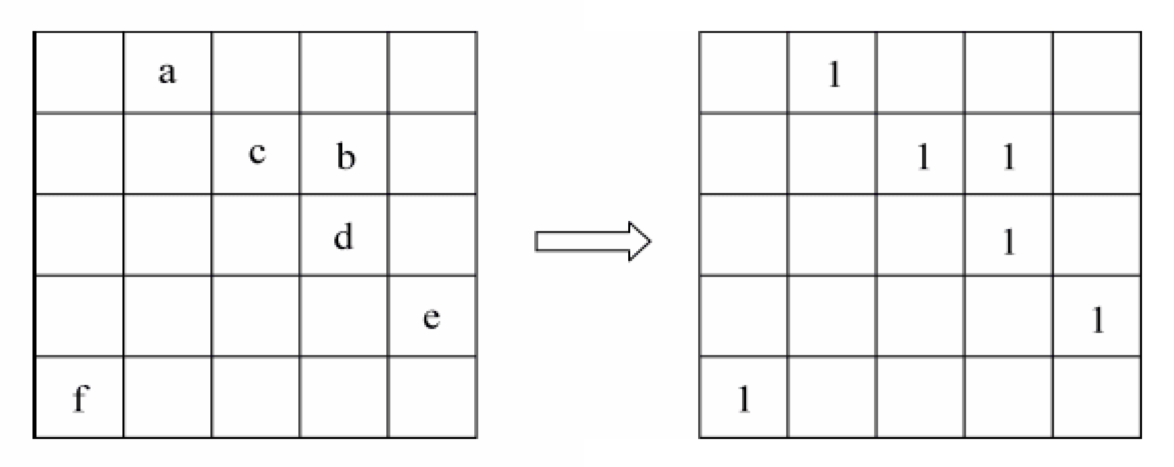


图5.6　控制流图矩阵转化为连接矩阵

图5.6中含两个或两个以上项的行，表示此行含有判定结点，判定结点的数目为行数，这为计算环形复杂性提供了一种方法。

5.3.2　基路径测试

程序的所有路径作为一个集合，在这些路径集合中必然存在一个最短路径，这个最短的路径称为基路径或独立路径。基路径测试是在程序控制流图的基础上，通过分析控制构造环路复杂性，确定基路径，设计测试用例覆盖这些基路径。

基本路径测试法主要步骤如下所示。

① 以详细设计或源代码作为基础，导出程序的控制流图。

② 计算控制流图 G 的圈复杂性 $V(G)$。

圈复杂度 $V(G)$ 为程序逻辑复杂性提供定量的测度，该度量用于计算程序的基本独立路径数目，确保所有语句至少执行一次的测试数量的上界。

③ 确定独立路径的集合，即确定线性无关的路径的基本集。

独立路径是指至少引入程序的一个新处理语句集合或一个新条件的路径，即独立路径必须包含一条在定义之前不曾使用的边。

④ 测试用例生成，确保基本路径集中每条路径的执行。

下面，介绍几种方法计算控制流图的圈复杂度 $V(G)$。

方法1：圈复杂度 $V(G)=E-N+2$，E 是流图中边的数量，N 是流图中结点的数量。

如图5.7所示，使用公式 $V(G)=E-N+2$，其中 E 为10，N 为7，则 $V(G)=10-7+2=5$，则圈复杂度为5。

方法2：圈复杂度 $V(G)$ 为控制流图中的区域数。

图5.7中的区域数为5，故，$V(G)=5$。

方法 3：圈复杂度 $V(G)=P+1$，P 是流图 G 中判定(谓词)结点的数量。

图 5.7 中的判定结点为 A、B、C、D，即 $P=4$，则根据公式 $V(G)=P+1=4+1=5$。

方法 4：从控制流图 5.6 转化为连接矩阵，若图中某行含两个或两个以上项，则此行为一个判定结点。

图 5.7 转化为连接矩阵如图 5.8 所示，具有 7 行 7 列的连接矩阵，即判定结点为 4，故圈复杂度 $V(G)$ 为 5。

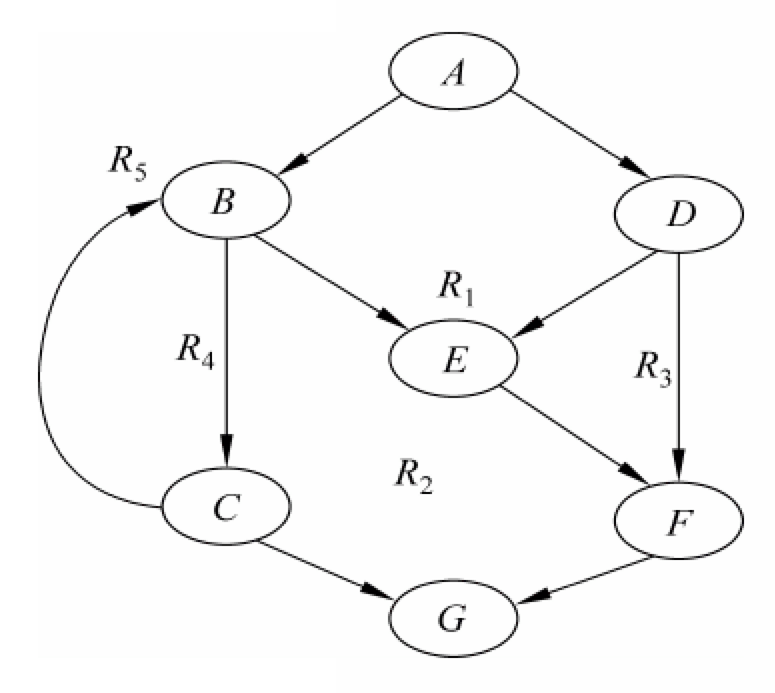


图 5.7　控制流图 G

	a	b	c	d	e	f	g
a		1		1			
b			1		1		
c		1					1
d					1	1	
e						1	
f							1
g							

图 5.8　连接矩阵

【例 5-3】 使用基路径测试方法测试以下程序段。

```
int  Sort ( int  iRecordNum, int iType )
1  {
2    int  x=0;
3    int  y=0;
4    while ( iRecordNum-->0)
5    {
6      If ( iType==0 )
7      x=y+2;
8      else
9      If ( iType==1 )
10        x=y+10;
11      else
12        x=y+20;
13    }
14  Return x }
```

【解答】 具体步骤如下所示。

① 此程序段的程序流程图如图 5.9 所示。

② 画出控制流图，如图 5.10 所示。计算圈复杂度 $V(G)$，导出独立路径。

其中，A 代码 5、6；B 代表 8、9；C 代表 11、12。

圈复杂度

$$V(G)=10(\text{条边})-8(\text{个结点})+1=4$$

独立路径：

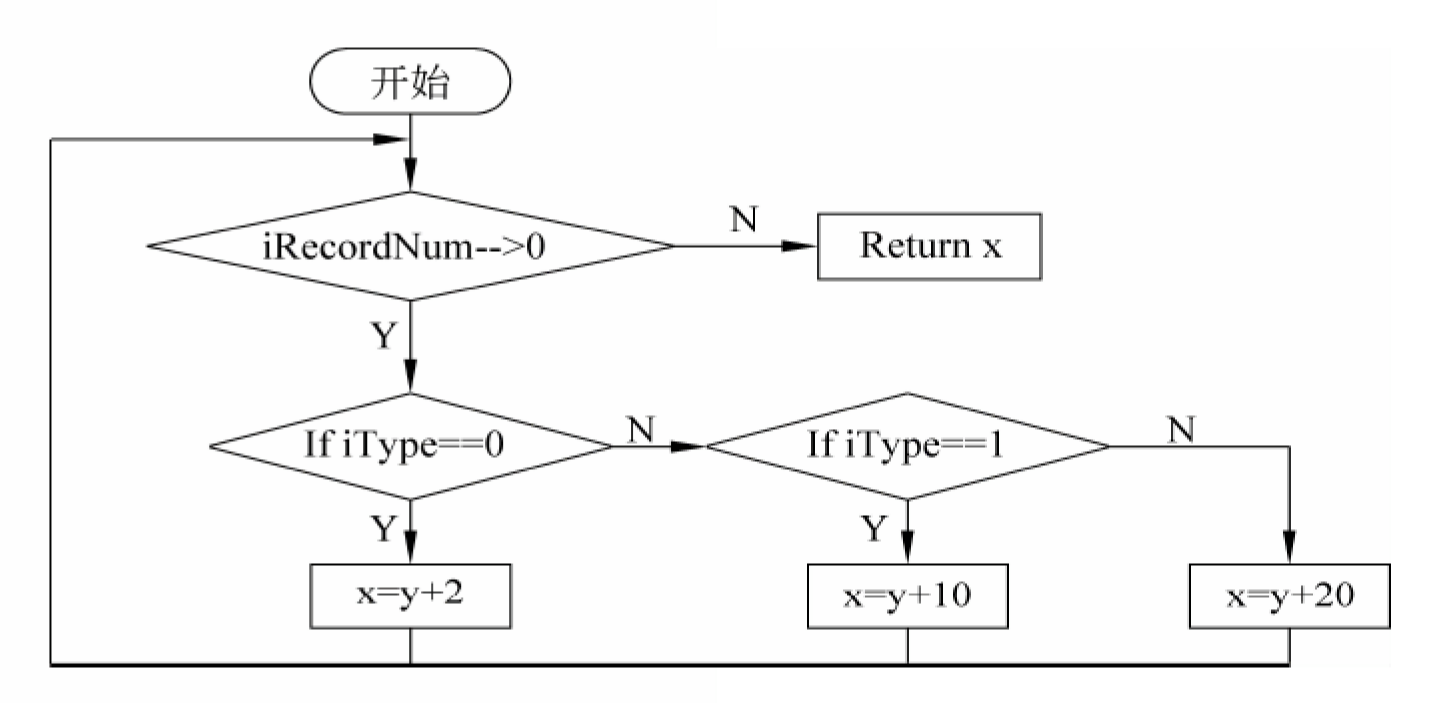


图 5.9　程序流程图

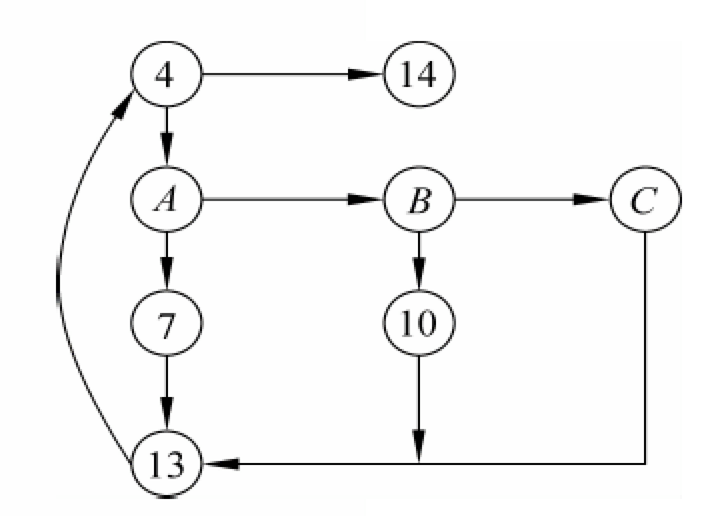


图 5.10　控制流图的基本符号示意图

Path1：4→14　　（判定结点在 4 处转向）
Path2：4→A→7→13→4→14
Path3：4→A→B→10→13→4→14　　（判定结点在 A 处转向）
Path4：4→A→B→C→13→4→14　　（判定结点在 B 处转向）

5.3.3　循环测试

循环结构是程序设计中运用最多的基本结构，由循环体及循环控制条件两部分组成。一般有简单循环、串接循环、嵌套循环等，如图 5.11 所示。

1. 简单循环

简单循环如图 5.11(a)和图 5.11(b)所示。考虑循环次数的边界值和接近边界值的情况，一般需要考虑如下几种测试用例，假设 n 是允许通过循环的最大次数。

(1) 零次循环：从循环入口直接跳到循环出口。
(2) 一次循环：只有一次通过循环，用于查找循环初始值方面的错误。
(3) 二次循环：两次通过循环，用于查找循环初始值方面的错误。
(4) m 次循环：m 次通过循环，其中 $m<n$，用于检查在多次循环时才能暴露的错误。
(5) 比最大循环次数少一次：即 $n-1$ 次通过循环。
(6) 最大循环次数：n 次通过循环。
(7) 比最大循环次数多一次：$n+1$ 次通过循环。

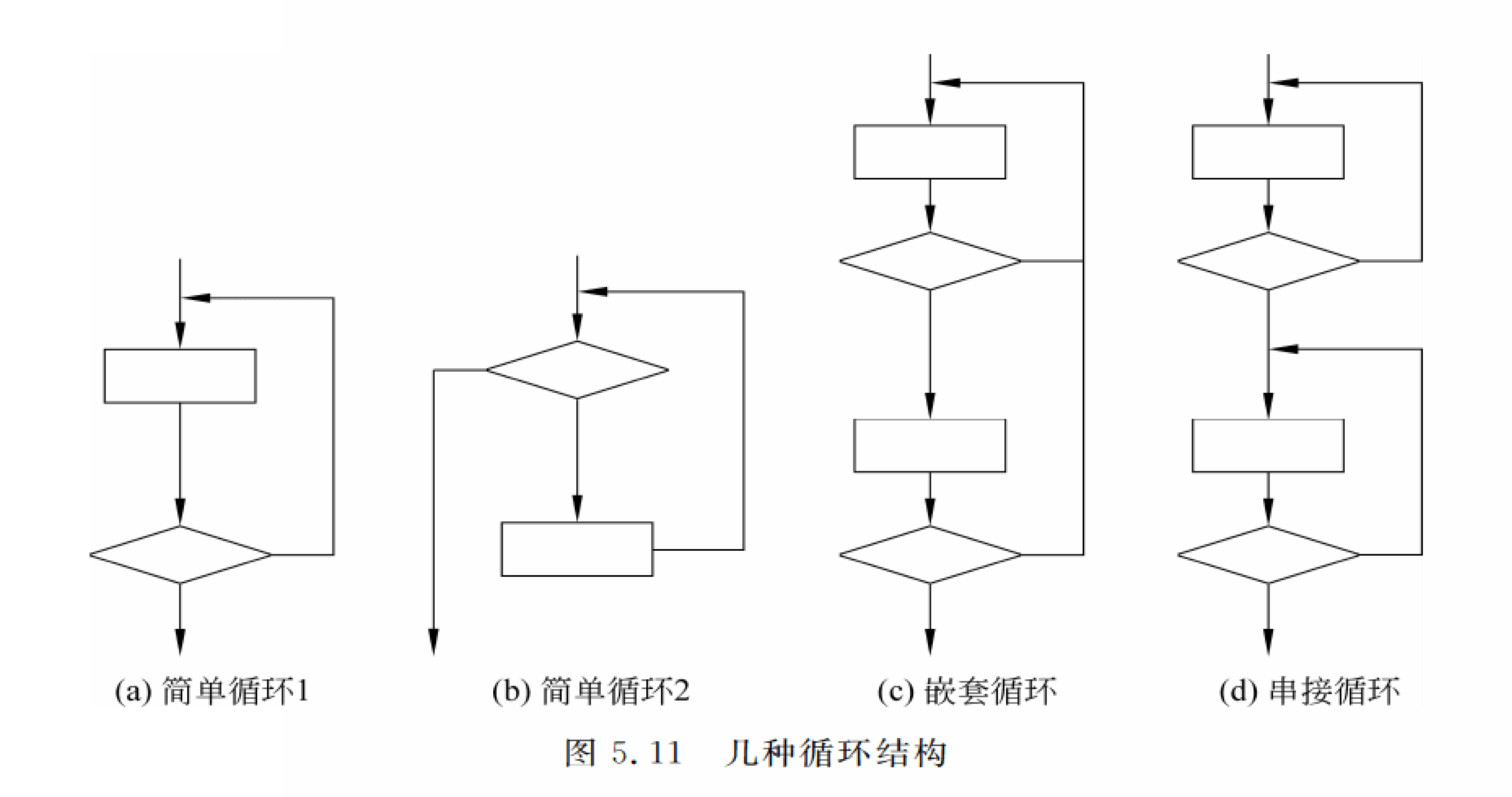


图 5.11 几种循环结构

2. 嵌套循环

嵌套循环如图 5.11(c)所示。如果要将简单循环的测试方法用于嵌套循环,可能的测试数就会随嵌套层数成几何级增加。

Beizer 提出了如下的减少测试数目的方法:

(1) 从最内层循环开始,将其他循环设置为最小值。

(2) 对最内层循环使用简单循环测试,而使外层循环的迭代参数(即循环计数)最小,并为范围外或排除的值增加其他测试。

(3) 由内向外构造下一个循环的测试,但其他的外层循环为最小值,并使其他的嵌套循环为"典型"值。

(4) 反复进行,继续直到测试完所有的循环。

3. 串接循环

串接循环又名并列循环,如图 5.11(d)所示。如果串接循环的循环都彼此独立,可以简化为两个单个循环来分布处理。但是,如果两个循环串接起来,而第一个循环的循环计数是第二个循环的初始值,则这两个循环并不是独立的。如果循环不独立,则应采用嵌套循环的方法进行测试。

5.3.4 逻辑覆盖法与路径测试比较

逻辑覆盖方法中有语句覆盖、判定覆盖、条件覆盖、判定-条件覆盖、条件组合覆盖等,其中语句覆盖最弱,依次增强,路径覆盖功能最为强大,包含着逻辑覆盖的各种覆盖方法,如图 5.12 所示。

表 5.13 给出了逻辑覆盖各种方法与基路径测试的各自优缺点。在实际测试中,往往根据测试用例设计的需要,将不同的设计方法有效地结合起来,设计出覆盖率最大、最有效的测试用例。

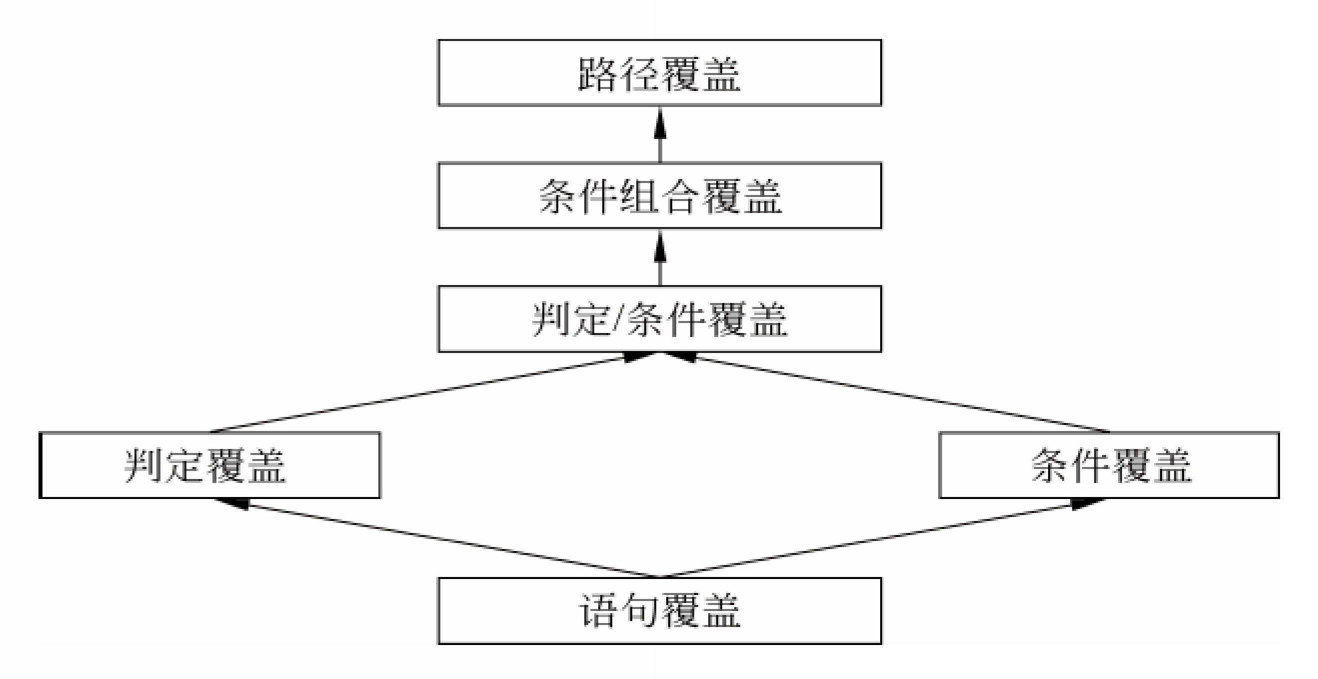


图 5.12　逻辑覆盖法总结

表 5.13　逻辑覆盖法对比

方法	判定覆盖	条件覆盖	条件组合覆盖	基路径测试
优点	简单、无须细分每个判定	增加了对条件判定情况的测试	对程序进行较彻底的测试，覆盖面广	测试用例清晰、有效
缺点	往往大部分的判定语句是由多个逻辑条件组合而成(如包含 AND、OR 等的组合)，若仅仅判断其组合条件的结果，而忽略每个条件的取值情况，必然会遗漏部分测试场景	达到条件覆盖，需要足够多的测试用例，但条件覆盖还是不能保证判定覆盖，这是由于 AND 和 OR 不同的组合效果造成的	对所有可能条件进行测试，需要设计大量、复杂测试用例 工作量比较大	基本路径法，类似于分支的方法，不能覆盖一些特定的条件，这些条件往往是容易出错的地方

5.4　数据流测试

数据流测试是从变量定义点与引用点的内在关系出发，利用了变量之间的关系进行的结构性测试。

早期的数据流测试分析常常集中于定义/引用异常的缺陷，用于如下三方面测试。

(1) 变量被定义，但是从来没有使用(引用)。

(2) 所使用的变量没有被定义。

(3) 变量在使用之前被定义两次。

从中可以得出，早期的数据流测试主要用于检测程序编写时出现的一些警告信息，如“所定义的变量未被使用等”问题，这些问题光靠简单的语法分析器或者是语义分析器是无法检测出来的。

数据流测试具有两种方法，一种称为变量定义/引用分析法，另一种称为程序片法。下面依次进行介绍。

5.4.1　变量定义/引用分析

下面介绍几个与变量定义/引用分析相关的概念。

1. DEF(v,n)

变量 v 在结点 n 处定义，定义包括输入语句，赋值语句(等号左侧)，过程调用都是定义结点的例子，如果执行这些语句，变量的值往往会发生变化。

2. USE(v,n)

变量 v 在结点 n 处被使用，使用包括输出语句、赋值语句(等号右侧)、条件语句、循环语句、过程调用语句都是结点的使用语句，如果执行这类语句，值不会被改变。

3. P-use

当且仅当 USE(v,n)是谓词使用，例如 a>=2，则 P-use 的程序图出度大于等于 2。

4. C-use

当且仅当 USE(v,n)是计算使用，则 C-use 的程序图入度小于等于 1。

5. **定义使用路径**

关于变量 v 的定义使用路径(记做 du-path)，存在定义和使用结点 DEF(v,m) 和 USE(v,n)，使得 m 和 n 是该路径的开始结点和结束结点。

6. **清除路径**

当定义结点和清除结点中间没有其他的定义结点的时候为清除路径。

【例 5-4】 数据流测试举例。

【解析】 图 5.13 是一个程序的控制流图，每个语句中变量的定义/引用由表 5.14 给出。表 5.14 中可知，语句 1 定义了变量 X,Y,Z，表明 3 个变量是被定义变量，语句 10 为被引用变量 Z。

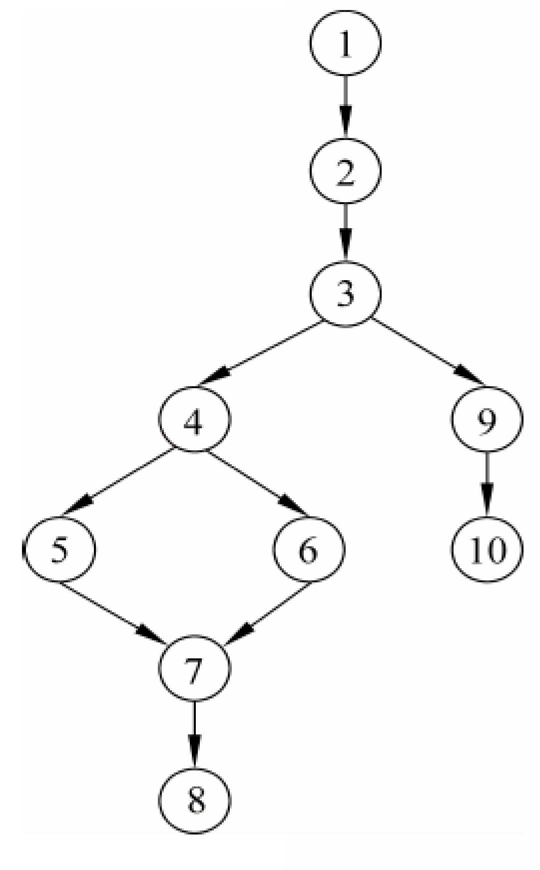


图 5.13 程序的控制流图

表 5.14 图 5.13 中的变量定义/引用

结点	被定义的变量	被引用的变量
1	X,Y,Z	
2	X	W,X
3		X,Y
4		Y,Z
5		V,Y
6		V,Z
7	V	X
8	W	Y
9		V
10		Z

通过变量的定义/引用分析，得出该程序中含有几个数据流异常：

(1) 语句 2 使用了变量 W，而在此之前并未对其进行定义（赋值）。

(2) 语句 5，6 使用了变量 V，但未对其定义过。

(3) 语句 8 对变量 W 的定义从未被使用过。

5.4.2 程序片

首先，介绍程序片的一个重要概念——S(V，n)。S(V，n)表示结点 n 之前对 V 的变量值做出贡献的所有语句片段的总和。

程序片往往用来排除程序的某些无意义的片段。比如有两个程序片，一个是第 8 行的 v，另一个是第 10 行的 v，假设第 8 行和第 10 行之间没有对 v 进行赋值，那么 p2＝(p1，9，10)，这里假设第 9 行和第 10 行影响了 v 的值，而若 8 行之前的程序片 p1 中的变量 v 没有发生问题，第 10 行的程序片 p2 出现了问题，必然变量 v 的异常必然在 p2－p1 这段程序片上。

数据流测试往往应用于计算密集的程序，数据流能够很方便地描述程序的部分片段结构。而程序片通常不能很好地设计测试用例，这是由于程序片是反映局部状况的，而由于变量的定义/引用分析是基于路径的，具有结构化的全局特性，故能反映出程序代码的某些异常来。

5.5 程序插桩

程序插桩是指通过借助于往被测程序中插入或添加一些语句，实现对程序语句的执行、变量的变化等情况进行测试。

在程序的特定部位插入记录动态特性的语句，把程序执行过程中发生的一些重要事件记录下来。例如，记录在程序执行过程中某些变量值的变化情况，变化的范围等。这些插入的语句常常被称为“探测器”或者“探测点”。设计“探测点”一般需要考虑如下问题：

- 探测哪些信息。
- 在程序的什么部位设置探测点。
- 需要设置多少个探测点。

程序插桩需要从插桩位置、插桩策略、插桩过程等方面进行考虑，下面依次进行介绍。

1. 插桩位置

插桩位置主要解决的是在哪儿插，为此将程序按“块”划分，探针主要插桩在其“路口”的位置，如下所示。

(1) 程序的开始，即程序块的第 1 个可执行语句之前。

(2) 转移指令之前。

- for、do、do-while、do until 等循环语句处。

- if、else if、else 及 end if 等条件语句各分支处。
- 输入/输出语句之后。
- 函数、过程、子程序调用语句之后。

(3) 标号之前。

(4) 程序的出口。

- return 语句之后。
- call 语句之后。

2. 插桩策略

插桩策略主要解决的如何在程序中植入探针,包括植入的位置和方法,分为块探针和分支探针。

- 块探针设计策略:又称"顺序块",是若干个相连顺序语句的序列集合。若该线性块的第一条语句被执行,则整个线性块都语句都执行了。这样仅在线性块的开始或末尾处插入一个探针即可,避免了对每条语句都进行操作。
- 分支探针策略:所有进行 True 或 False 判断的语句处进行插桩。

3. 插桩过程

【例 5-5】 程序插桩举例。

【解析】 计算整数 X 和整数 Y 的最大公约数,图 5.14 给出了插桩后的最大公约数程序的程序流程图,其中虚线框为记录语句执行次数而插入的,其形式为:

$$C(i)=C(i)+1,\quad i=1,2,3,\cdots,6$$

程序从入口开始执行到出口结束,经过的计数语句记录下该程序点的执行次数。如果在程序的入口处插入了对计数器 C(i)初始化的语句,在出口处插入了打印这些计数器的语句,就构成了完整的插桩程序,它便能记录并输出在各个程序点上的语句的实际的执行次数。

软件测试中使用程序插桩主要用于如下 3 方面:

(1) 覆盖分析

程序插桩可以用来确定和估计有关程序结构元素被覆盖的程度,从而确定测试执行的充分性,设计更好的测试用例,提高测试覆盖率。

(2) 监控和断言

在程序特定部位插入某些用以判断变量特性的断言语句,以便证实程序运行时的某些特性,从而帮助排除故障。

(3) 查找数据流异常

数据流插桩可以记录每个变量的最大值和最小值,从而发现超出预计范围的情况,还可以发现引用未经初始化的变量,以及已定义过但未曾使用的变量等数据流异常。

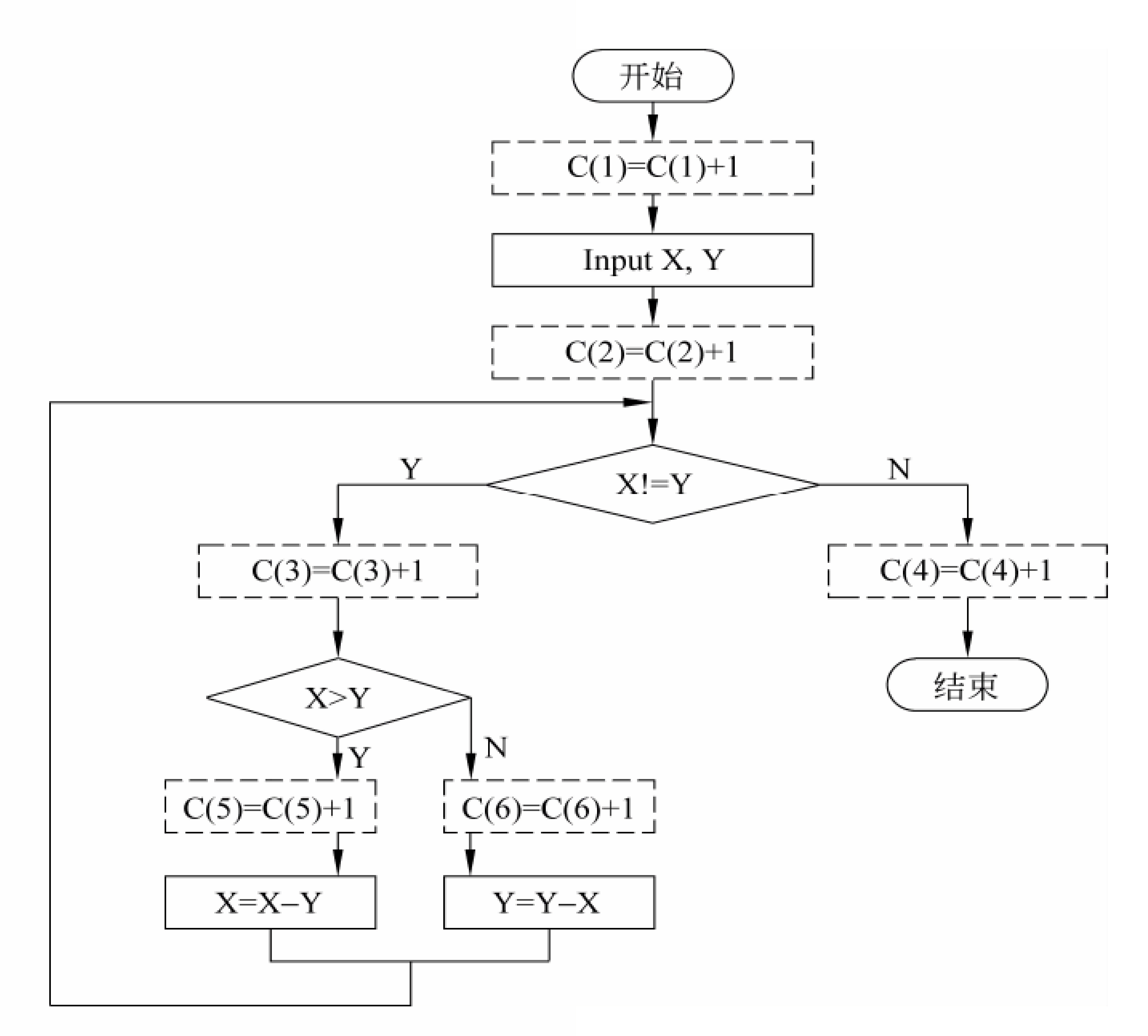
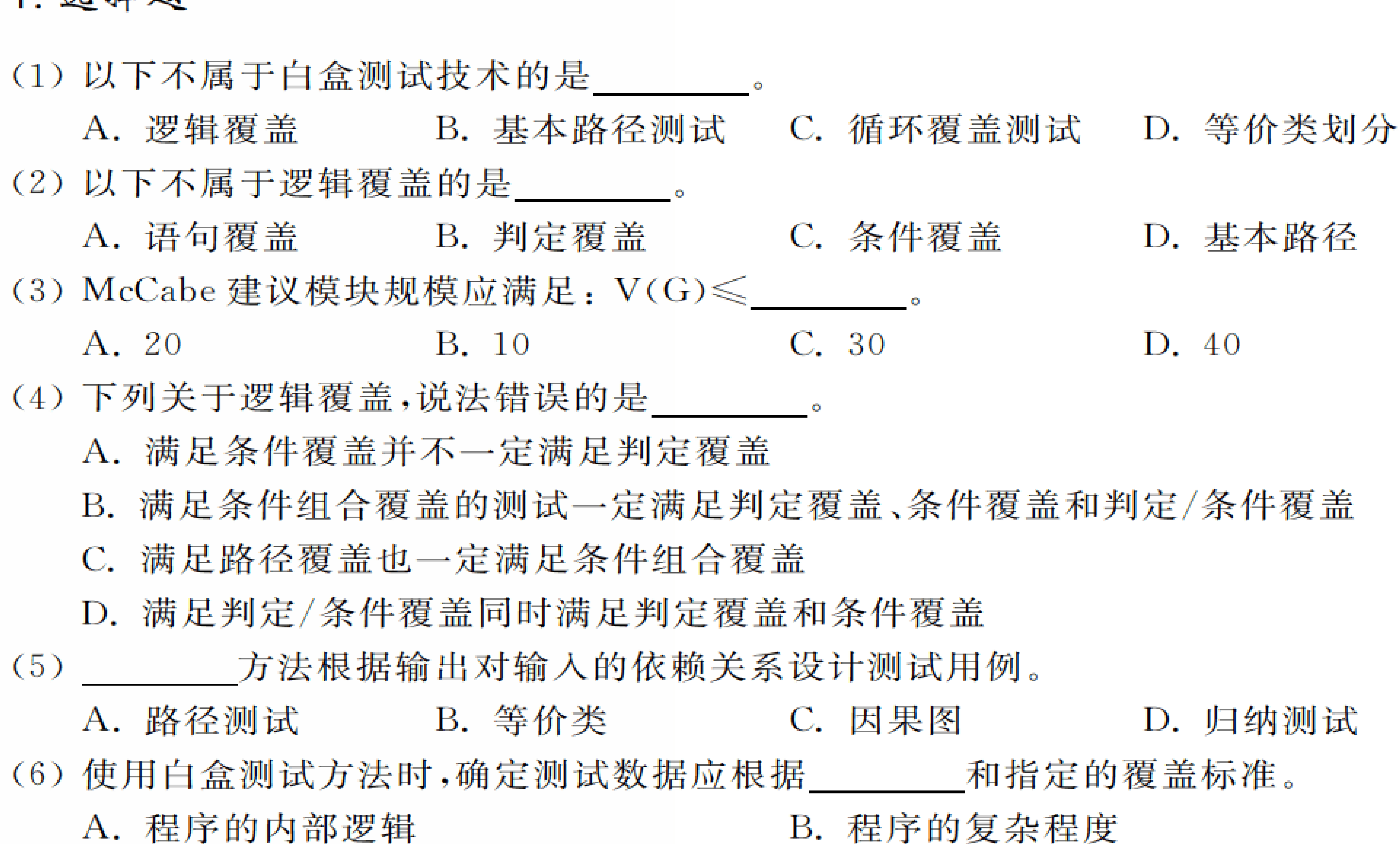


图 5.14　程序的程序流程图



5.6　习　　题

1. 选择题

(1) 以下不属于白盒测试技术的是________。

A. 逻辑覆盖　　B. 基本路径测试　　C. 循环覆盖测试　　D. 等价类划分

(2) 以下不属于逻辑覆盖的是________。

A. 语句覆盖　　B. 判定覆盖　　C. 条件覆盖　　D. 基本路径

(3) McCabe 建议模块规模应满足：V(G)≤________。

A. 20　　B. 10　　C. 30　　D. 40

(4) 下列关于逻辑覆盖，说法错误的是________。

A. 满足条件覆盖并不一定满足判定覆盖

B. 满足条件组合覆盖的测试一定满足判定覆盖、条件覆盖和判定/条件覆盖

C. 满足路径覆盖也一定满足条件组合覆盖

D. 满足判定/条件覆盖同时满足判定覆盖和条件覆盖

(5) ________方法根据输出对输入的依赖关系设计测试用例。

A. 路径测试　　B. 等价类　　C. 因果图　　D. 归纳测试

(6) 使用白盒测试方法时，确定测试数据应根据________和指定的覆盖标准。

A. 程序的内部逻辑　　B. 程序的复杂程度

C. 使用说明书　　D. 程序的功能

(7) 白盒测试方法的优点是________。

A. 可测试软件的特定部位　　B. 能站在用户立场上测试

C. 可按软件内部结构测试　　D. 可发现实现功能需求中的错误

2. 简答题

(1) 白盒测试是什么? 和黑盒测试的区别体现在哪些方面?

(2) 为什么说语句覆盖是最弱的逻辑覆盖?

(3) 条件覆盖为什么不一定包含判定覆盖?

(4) 采用白盒法进行测试时,测试用例覆盖路径的种类有哪几种? 它们相互之间是什么关系?

(5) 请把图 5.15 所示的程序流程图转化成控制流图。

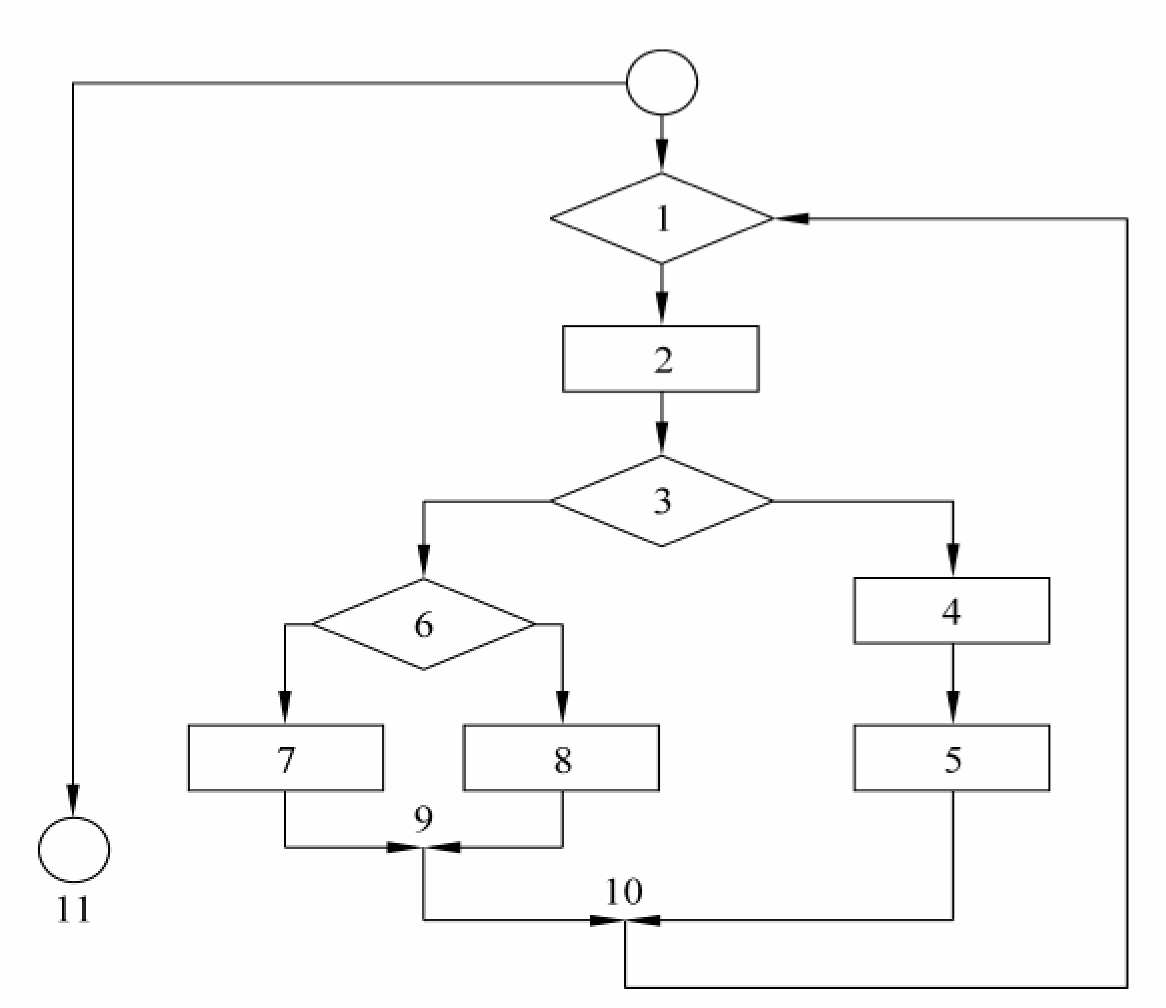


图 5.15　程序的程序流程图



性能测试

性能测试是通过自动化的测试工具模拟多种正常、峰值以及异常负载条件来对系统的各项性能指标进行测试。本章介绍了性能测试的基本概念、性能测试分类，包括负载测试、压力测试、可靠性测试、安全性测试、文档测试等，并就网站测试给出了详细的解说。

6.1 基本概念

性能是指在一定条件下系统的行为表现是否符合需求规格的指标数值，如传输的最长时限、传输的错误率、计算的精度、响应的时限和恢复时限等指标。性能测试就是验证软件系统是否能够达到需求规格说明中所提出的性能指标，同时发现软件系统中存在的性能瓶颈，优化软件系统。

性能测试主要包括以下几个方面：

1. 评估系统的能力

评估系统的能力是指测试软件系统所得到的负荷数据和响应时间等数据，用于验证软件系统的稳定性和可靠性。

2. 识别体系中的弱点

通过将软件系统受控的负荷增加到一个极端的水平，确定体系的瓶颈或薄弱的地方，并进行修复。

3. 系统调优

系统长时间的运行会导致系统失败，揭示系统中的隐含问题或冲突，需要进行调整，优化系统性能。

下面，介绍一些常见的性能指标，如响应时间、并发用户数、性能计数器、休眠时间等。

1. 响应时间

响应时间是指“对请求做出响应所需要的时间”，分解为网络传输时间、应用延迟时间、数据库延迟时间和呈现时间等。其中，“呈现时间”取决于数据在被客户端收到数据后呈现页面所消耗的时间，“系统响应时间”指应用系统从请求发出开始到客户端接收到数

据所消耗的时间。一般而言,网站的响应时间有2秒、5秒和10秒几个标准。2秒之内响应客户被认为是“非常有吸引力的”,5秒之内响应客户认为是“比较不错的”,而10秒是客户能接受的响应的上限,也就是用户打开网页花费时间超过了10秒钟,用户往往无法容忍。

2. 并发用户数

多个用户对系统发出了请求或者进行了操作,其请求或者操作可以是相同的,也可以是不同的。对整个系统而言,仍然是有多个用户同时对系统进行操作。下面给出估算并发用户数的公式。

$$C = \frac{nL}{T} \tag{6.1}$$

$$\hat{C} \approx C + 3\sqrt{C} \tag{6.2}$$

式(6.1)中,C 是平均的并发用户数;n 是登录会话的数量;L 是登录会话的平均长度;T 指考察的时间段长度。

式(6.2)给出了并发用户数峰值的计算方式,其中,$\hat{C}$指并发用户数的峰值,C 由式(6.1)中得到,该公式是假设用户登录会话符合泊松分布而估算得到的。

【例 6-1】 一个软件系统每天大约有400用户访问。用户在一天之内有8小时内使用该系统,从登录到退出该系统的平均时间为4小时。

【解答】 根据式(6.1)和式(6.2),得到:

$$C = 400 \times 4/8 = 200$$

$$\hat{C} \approx 200 + 3 \times \sqrt{200} = 242$$

3. 吞吐量

吞吐量是指在一次性能测试过程中网络上传输数据量的总和。一般来说,吞吐量用请求数/秒或是页面数/秒来衡量。从业务角度分析,吞吐量用访问人数/天或者处理业务数/小时等衡量。吐量指标有如下两个作用。

(1) 协助设计性能测试场景,以及衡量性能测试场景是否达到了预期的设计目标。在设计性能测试场景时,吞吐量用于协助设计性能测试场景,根据估算吞吐量数据,测试场景的事务发生频率等。

(2)协助分析性能瓶颈。吞吐量是性能瓶颈的重要表现形式。因此,有针对性地测试吞吐量,尽快定位到性能瓶颈所在位置。

吞吐量和并发用户数之间存在一定的联系。计算公式如下:

$$F = \frac{N_{vu} \times R}{T} \tag{6.3}$$

式中,F 表示吞吐量;N_{vu}表示虚拟用户个数;R 表示每个虚拟用户发出的请求数量;T 表示性能测试所用的时间。

4. 性能计数器

性能计数器是描述服务器或操作系统性能的一些数据指标,具有“监控和分析”作用。

例如，Windows 系统的内存数、进程数、系统缓存等都是常见的性能计数器。与性能计数器相关的“资源利用率”，是指系统各种资源使用状况，资源利用率＝资源的实际使用/总的资源可用量。

在通常的情况下，资源利用率需要结合响应时间变化曲线、系统负载曲线等各种指标进行综合的分析。

5. 休眠时间

休眠时间又称为思考时间，是指用户请求的间隔时间。在交互式应用中，用户发出一个请求，等待一段时间，再发出下一个请求。因此，自动化测试模拟用户操作就必须在测试脚本中让各个操作间隔一段时间，在操作语句之间设置 Think 函数，实现两个操作之间的等待时间。

休眠时间与迭代次数、并发用户数和吞吐量之间存在一定关系。式(6.3)说明吞吐量是 VU 数量 N_{vu}、每个用户发出请求数 R 和时间 T 的函数。其中 R 可以用时间 T 和用户的思考时间 T_s 来计算，如下所示：

$$R = \frac{T}{T_s} \tag{6.4}$$

比较式(6.3)和式(6.4)可知，吞吐量与 N_{vu} 成正比，而与 T_s 成反比。在实际测试中，通过如下步骤计算休眠时间。

① 首先计算出系统的并发用户数。

② 统计出系统平均的吞吐量。

③ 统计出平均每个用户发出的请求数量。

④ 根据式(6.4)计算出思考时间。

当然，为了让性能测试场景更加符合实际情况，以计算思考时间为基准，在一定幅度内随机变动。

6. 点击率

点击率是指每秒钟用户向 Web 服务器提交的 HTTP 请求数，作为 Web 应用特有指标。Web 应用是“请求-响应”模式，即用户发出一次请求，服务器响应请求，处理后返回给用户。点击是 Web 应用能够处理的最小单位，点击率越大，对服务器的压力也就越大。需要注意的是，这里的点击并非指鼠标的一次单击操作，因为在一次单击操作中，客户端可能向服务器发出多个 HTTP 请求。

6.2 性能测试分类

下面介绍一些常见的性能测试，如负载测试、压力测试、可靠性测试、数据库测试、安全性测试和文档测试等。

6.2.1 负载测试

负载测试(Load Testing)是通过测试系统在资源超负荷情况下的表现，以发现设计

上的错误或验证系统的负载能力，评估测试对象在不同工作量条件下的性能行为，以及持续正常运行的能力。负载测试的目标是确定并确保系统在超出最大预期工作量的情况下仍能正常运行。此外，负载测试还要评估性能特征，例如，响应时间、事务处理速率和其他与时间相关的方面。

通过大量重复的行为、模拟不断增加的用户数量等方式观察不同负载下系统的响应时间和数据吞吐量、系统占用的资源(如 CPU、内存)等，检验系统特性，发现系统可能存在的性能瓶颈、内存泄漏等问题。

负载测试的加载方式，通常有如下几种。

1. 一次加载

一次性加载某个数量的用户，在预定的时间段内持续运行。例如，早晨上班的时间，访问网站或登录网站的时间非常集中，基本属于扁平负载模式。

2. 递增加载

有规律地逐渐增加用户，每几秒增加一些新用户，借助这种负载方式的测试，容易发现性能的拐点，即性能瓶颈。

3. 高低突变加载

某个时间用户数量很大，突然降到很低，过一段时间，又突然加到很高，反复几次。借助这种负载方式的测试，容易发现资源释放、内存泄露等问题。

4. 随机加载方式

由随机算法自动生成某个数量范围内变化的、动态的负载，这种方式可能是和实际情况最为接近的一种负载方式。虽然不容易模拟系统运行出现的瞬时高峰期，但可以模拟系统长时间的高位运行过程的状态。

6.2.2 压力测试

压力测试(Stress Test，也称强度测试)是在强负载(大数据量、大量并发用户等)下的测试，通过查看应用系统在峰值使用情况下的状态，发现系统的某项功能隐患、系统是否具有良好的容错能力和可恢复能力。压力测试涉及时间因素，用来测试那些负载不定的、或交互式的、实时的以及过程控制等程序。压力测试分为高负载下的长时间(如 24 小时以上)的稳定性压力测试和极限负载情况下导致系统崩溃的破坏性压力测试。

压力测试也被看做是负载测试的一种特殊情况，即高负载下的负载测试，或者说压力测试采用负载测试技术。通过压力测试，往往可以发现影响系统稳定性的问题。例如，在正常负载情况下，某些功能不能正常使用或系统出错的概率比较低，可能一个月只出现一次，但在高负载(压力测试)下，可能一天就出现，从而发现缺陷。

微软测试实践经验表明，如果软件产品通过 72 小时压力测试，则在 72 小时后出现问题的可能性微乎其微。所以，72 小时成为微软产品压力测试时间标志。压力测试用例的

参考模板如图 6.1 所示。

1. 被测试对象的介绍
2. 测试范围与目的
3. 测试环境与测试辅助工具的描述
4. 测试驱动程序的设计
5. 压力测试用例

极限名称A	如“最大并发用户数量”	
前提条件		
输入/动作	输出/响应	是否能正常运行
如10个用户并发操作		
如20个用户并发操作		

图 6.1 压力测试用例的参考模板

压力测试检查系统在资源超负荷的情况下的表现，压力测试的其中一个变种——敏感测试，是指在有些情况下，数据界限内的很小范围的数据可能会引起错误的运行，或引起性能急剧下降，敏感测试用于发现可能会引起不稳定或错误处理的数据组合。

压力测试的主要目的是度量应用系统的性能和扩展性。在实施并发负载过程中，通过实时性能监测来确认和查找问题，并针对所发现问题对系统性能进行优化。压力测试工具能够对整个企业架构进行测试，通过这些测试，企业能最大限度地缩短测试时间，优化性能和加速应用系统的发布周期。

由于负载测试、压力测试和性能测试往往在测试手段和方法上比较相似，通常会使用相同的测试环境和测试工具，而且都会监控系统所占用资源的情况以及其他相应的性能指标，容易产生混淆。但是，性能测试、负载测试盒和压力测试三者的测试目的是不同的。性能测试用于检验某功能——模块或软件产品整体的工作效率，包括资源占用率、执行响应时间、B/S 或 C/S 结构的还涉及用户并发能力等。例如，检验不同数量用户登录访问网站时，服务器的承受能力，能支持多少用户并发访问，用户进行操作时服务器系统资源占用情况，不同用户并发操作过程单一业务执行响应时间等。性能测试是为了获得系统在某种特定的条件下的性能指标数据，而负载测试、压力测试是为了发现软件系统中所存在的问题，如内存泄漏等。

当通过负载测试为了获得系统正常工作时所能承受的最大负载，这时负载测试就成为容量测试。负载测试与压力测试的区别：负载测试是通过逐步增加系统的复杂性，观察其变化，看最后在满足性能的情况下，系统最多能接受多大负载的测试。压力测试可以被看作是负载测试的一种，就是发现在什么条件下系统的性能会变得不可接受。

负载测试与性能测试区分如下，负载测试是为了发现系统的性能问题，负载测试需要通过系统性能特性或行为来发现问题，从而为性能改进提供帮助，从这个意义看，负载测试可以看做性能测试的一部分。但它们两者的目的是不一样的，负载测试是为了发现缺陷，只测试在一些极端条件下，系统还能否正常工作，或加载到系统崩溃而找出系统性能

的瓶颈。而性能测试是为了获取性能指标。因为性能测试过程中,也可以不调整负载,而是在同样负载情况下改变系统的结构、改变算法、改变硬件配置等等来得到性能指标数据,从这个意义看,负载测试可以看做是性能测试的一种技术,即性能测试使用负载测试的技术、使用负载测试的工具。性能测试要获得在不同的负载情况下的性能指标数据。

6.2.3 可靠性测试

软件可靠性是软件质量的一个重要标志。IEEE将软件可靠性定义为:系统在特定的环境下,在给定的时间内无故障地运行的概率。软件可靠性涉及软件的性能、功能性、可用性、可服务性、可安装性,可维护性等多方面特性,是对软件在设计、生产以及在它所预定环境中具有所需功能的可信度的一个度量。

可靠性测试一般伴随着强壮性测试,是评估软件在运行时的可靠性,通过测试确认平均无故障时间、故障发生前平均工作时间或因故障而停机的时间在一年中应不超过多少时间。可靠性测试强调随机输入,并通过模拟系统实现,很难通过实际系统的运行来实现。

6.2.4 数据库测试

数据库测试一般包括数据库的完整测试和数据库容量测试。下面依次介绍。

1. 数据库完整测试

数据库完整测试是指测试关系型数据库完整性原则以及数据合理性测试。数据库完整性原则是指:

(1) 主码完整性:主码不能为空。

(2) 外码完整性:外码必须等于对应的主码或者为空。

(3) 用户自定义完整性。

例如性别字段的取值只能是"男"或者"女"。某信息管理系统有两张表:部门表和员工表。其中,部门表有部门编号、部门名称、部门经理等字段,主码为部门编号;员工表中有员工编号、员工所属部门编号、员工名称、员工类型等字段,主码为员工编号,外码为员工所属部门编号,对应部门表。如果在某条部门记录中部门编号或员工记录员工编号为空,就违反"主码完整性"原则。如果某个员工所属部门的编号为##,但是##在部门编号中却找不到,就违反了"外码完整性"的原则。又例如,通过用户自定义完整性将员工表的年龄属性限制在20~35岁之间,如果用户输入的年龄不在这个范围之内,就违反了"用户自定义完整性"的原则。

数据合理性指数据在数据库中的类型、长度、索引等是否建的比较合理。在项目名称中,数据库和数据库进程应作为一个子系统来进行测试。

2. 数据库容量测试

数据库容量测试指通过存储过程往数据库表中插入一定数量的数据,看看相关页面是否能够及时显示数据。数据库容量测试使测试对象处理大量的数据,以确定是否达到

了将使软件发生故障的极限。容量测试还将确定测试对象在给定时间内能够持续处理的最大负载或工作量。例如，往员工表中插入高数量级的数据，看其是否可以正常显示顾客信息列表页面，是否发生异常。

6.2.5 安全性测试

安全性测试是测试系统在应付非授权的内部/外部访问、非法侵入或故意的损坏时的系统防护能力，检验系统有能力使可能存在的内/外部的伤害或损害的风险限制在可接受的水平内。可靠性通常包括安全性，但是软件的可靠性不能完全取代软件的安全性，安全性还涉及数据加密、保密、存取权限等多个方面。

安全性测试时需要设计一些测试用例试图突破系统的安全保密措施，检验系统是否有安全保密漏洞，验证系统的保护机制是否能够在实际中不受到非法的侵入。安全性测试采用建立整体的威胁模型，测试溢出漏洞、信息泄漏、错误处理、SQL 注入、身份验证和授权错误、XSS 攻击。在安全测试过程中，测试者扮演成试图攻击系统的角色设计测试用例。例如：

(1) 尝试截取、破译、获取系统密码。

(2) 让系统失效、瘫痪，控制系统，使他人无法访问，自己非法进入。

(3) 试图浏览保密的数据，检验系统是否有安全保密的漏洞。

6.2.6 文档测试

软件文档有助于程序员编制程序，有助于管理人员监督和管理软件的开发，有助于维护人员进行有效的修改和扩充等，因此，软件文档具有针对性、精确性、清晰性、完整性、灵活性和可追溯性等特性。

(1) 针对性：文档编制应分清读者对象。按不同的类型不同层次的读者进行区分。例如，管理文档主要是面向管理人员的，用户文档主要是面向用户的，这两类文档不应像面向开发人员的开发文档过多使用软件的专用术语。

(2) 精确性：文档的行文应当十分确切，不能出现多义性的描述。同一课题几个文档的内容应当协调一致，没有矛盾。

(3) 清晰性：文档编写应力求简明，如有可能，配以适当的图表。

(4) 完整性：任何一个文档都应当是完整的、独立的，自成体系。例如，前言部分应做一般性介绍，正文给出中心内容，必要时还有附录，列出参考资料等。同一课题的几个文档之间可能有些部分内容相同，这种重复是必要的，不要在文档中出现转引其他文档内容的情况。例如，段落描述用“见××文档××节”的方式会给读者带来许多的不便。

(5) 灵活性：各个不同软件项目，其规模和复杂程度有着许多实际差别。

① 根据具体的软件开发项目，决定编制的文档种类。

软件开发的管理部门应该根据本部门承担的应用软件的专业领域，制定一个对文档编制要求的实施规定。

对于一个具体的应用软件项目，项目负责人应根据上述实施规定，确定一个文档编制计划。其中包括：应该编制哪几种文档，详细程度如何；各个文档的编制负责人和进度要

求；审查、批准的负责人和时间进度安排；在开发时期内各文档的维护、修改和管理的负责人，以及批准手续。

② 软件系统非常大时，文档可以分成几卷编写。

根据项目的开发与测试阶段不同，文档有如下分类。项目开发计划可分为：质量保证计划、配置管理计划、用户培训计划、安装实施计划等。系统设计说明书可分为：系统设计说明书、子系统设计说明书。程序设计说明书可分为：程序设计说明书、接口设计说明书、版本说明。操作手册可分为：操作手册、安装实施过程。测试计划可分为：测试计划、测试设计说明、测试规程、测试用例。测试分析报告可分为：综合测试报告、验收测试报告。项目开发总结报告也可分成：项目开发总结报告、资源环境统计。

③ 根据任务的规模、复杂性、项目负责人对该软件的开发过程及运行环境所需详细程度的判断，确定文档的详细程度。

④ 文档内容根据《计算机软件测试文件编制指南》指导进行，其中关于所建议的所有条款都可以扩展，进一步细分；反之，如果条款中有些细节并非必需，也可以根据实际情况进行压缩合并。

⑤ 程序设计表现形式可以使用程序流程图、判定表、程序描述语言和问题分析图等。

（6）可追溯性：由于各开发阶段编制的文档与各个阶段完成的工作有密切的关系，文档具有一定的继承关系，项目各个开发阶段之间提供的文档必定存在着可追溯的关系。例如，系统的软件需求，必定在设计说明书、测试计划，甚至用户手册中有所体现。

6.3 性能测试的步骤

针对不同的系统架构，开发人员可能选择不同的实现方式。下面介绍一种关于如何选择测试策略的方法，帮助分析软件系统的整体架构的性能指标和性能瓶颈，其步骤如下所示：

① 制定目标和分析系统。
② 选择测试度量的方法。
③ 采用相关技术和工具。
④ 制定评估标准。
⑤ 设计测试用例。
⑥ 运行测试用例。
⑦ 分析测试结果。

1. 制定目标和分析系统

性能测试计划中第一步都会制定目标和分析系统。只有明确目标和了解系统构成才会澄清测试范围，知道在测试中要掌握什么样的技术。明确目标是指确定客户需求和期望、实际业务需求和系统需求。

系统组成明确测试的范围，选者适当的测试方法来进行测试。系统组成包含系统类别、系统构成、系统功能等。系统类别采用都体系结构是 B/S 结构，需要掌握 HTTP 协

议、Java、HTML 等技术；若系统为 C/S 结构，需要了解 OS、Winsock 等。不同的系统构成性能测试就会得到不同的结果。一般性能测试都是利用测试工具模仿大量的实际用户操作，系统在超负荷情形下运行。系统功能是性能测试中要模拟的环节，是指系统提供的不同子系统、办公管理系统中的公文子系统、会议子系统等。

2. 选择测试度量方法

经过第一步的制定目标和分析系统后，接下来进行软件度量，收集系统相关的数据。度量的相关方面具有如下内容：

- 制定规范。
- 制定相关流程、角色、职责。
- 制定改进策略。
- 制定结果对比标准。

3. 采用相关技术和工具

性能测试是通过测试工具，模拟大量用户操作，对系统增加负载，所以必须熟练地掌握和运用测试工具。由于性能测试工具一般基于不同的软件系统架构实现，其脚本语言也不同，只有经过工具评估，才能选择符合现有软件架构的性能测试工具，确定测试工具后，需要组织测试人员进行工具的学习，培训相关的测试技术。

4. 制定评估标准

任何测试的目的是确保软件符合预先规定的目标和要求。通常性能测试有线性投射、分析模型、模仿和基准 4 种模型技术用于评估。

1）线性投射

通过大量的过去的，扩展的或者将来可能发生的数据组成散布图，利用这个图表不断和系统的当前状况进行对比。

2）分析模型

通过预测响应时间，将工作量的数据和系统本质关联起来，进行分析模型。

3）模仿

模仿实际用户的使用方法反复的测试系统

4）基准

定义测试作为标准，与后面进行的测试结果进行对比

5. 设计测试用例

设计测试用例的原则是受最小的影响提供最多的测试信息，设计测试用例的目标是一次尽可能的包含多个测试要素，这些测试用例必须是测试工具可以实现的，不同的测试场景将测试不同的功能。

6. 运行测试用例

通过性能测试工具运行测试用例，需要不同的测试环境，以及不同的机器配置。

7. 分析测试结果

运行测试用例后，收集相关信息，进行数据统计分析，找到性能瓶颈。通过排除误差和其他因素，让测试结果体现真实情况。不同的体系结构分析测试结果的方法也不同，B/S 结构的系统通常会分析网络带宽，流量对用户操作响应的影响，而 C/S 结构可能更关心系统整体配置对用户操作的影响。

6.4 网站测试

网站作为基于 Web 的软件架构，其测试与传统的软件测试不同，不但需要检查和验证网站是否按照设计的要求运行，还要测试网站是否适合不同用户的浏览器显示，并要从最终的使用用户的角度进行安全性和可用性的各项测试。

6.4.1 网站结构模型

网站属于客户/服务器软件类别，相对于一般的窗口软件有其明显的特点，如图 6.2 所示，网站的结构模型由客户层、表示层、逻辑业务层和数据层等几部分组成。

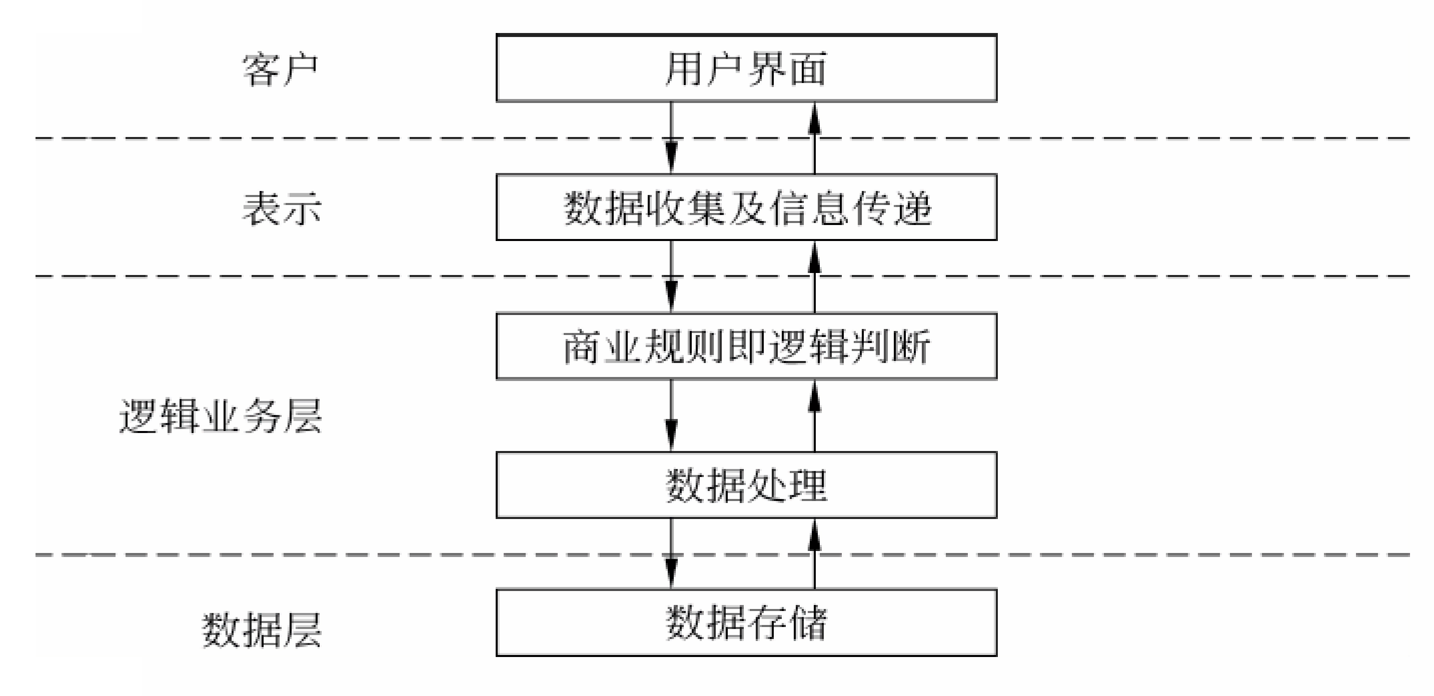


图 6.2 网站的结构模型

(1) 第一层是客户层，主要提供用户界面，作为使用者交互操作平台，用于数据输入和信息获取。

(2) 第二层是表示层，负责将输入数据进行处理，送到下一层。同时，也负责将下一层传回的数据显示在用户界面。

(3) 第三层是逻辑业务层，负责将上一层传来的数据按照需求规格中特定的业务规则及设定的逻辑进行处理，传送到数据存储层。

(4) 第四层是数据层，一般使用数据库作为数据的存储与管理。

6.4.2 网站测试内容

在网站测试中主要进行以下项目的测试。

1. *用户界面测试*

测试页面是否美观，包括页面的布局是否合理，页面长度是否合理，前景色与背景色是否搭配，页面风格是否统一

2. *链接测试*

链接测试主要是测试点击链接时是否可以进入所找的页面，是否能正确返回，链接页面会不会是空白页面、孤立页面或根本没链接(也就是说链接的是自己本身)，如果链接的是空白页我们是否可以正确返回，如果使用了框架或内嵌框架是否可以在本框架内正确地显示要查找的页面，使用内容置顶时是否可以正确实现。

3. *表单测试*

表单的测试包括单选按钮、复选框、文本框、密码项和菜单项和提交按钮类按钮的测试和后台数据库的测试。

4. *兼容性测试*

在配置不同操作系统和不同分辨率的计算机上使用不同的浏览器对其测试，看其是否可以正确显示，是否有图片和页面错位等问题使有的部分无法看到，是否有图片或视频无法显示。

5. *网络配置测试*

网络配置测试网页是否可以保存页面，测试网页冗余代码是否过多或容量太大导致网络运行速度过慢。

6. *负载测试*

负载测试主要测试多个用户同时上网，其最大的承受能力是多大，如果超过了这个极限会有何反应。

7. *安全测试*

安全测试主要测试用户名和密码是否有长度限制，是否有复杂度限制，登录次数是否受限。

6.5 习　　题

1. *判断题*

(1) 负载测试是验证系统在资源超负荷情况下的表现。　　(　　)

（2）所有软件必须进行某种程度的兼容性测试。 （ ）
（3）以消除瓶颈为目的的测试是负载测试。 （ ）

2. 简答题

（1）请解释如下名词概念。
响应时间、并发用户数、吞吐量、性能计数器、休眠时间、点击率
（2）负载测试与压力测试有什么异同点？请举例说明。
（3）什么是可靠性测试？请举例说明。
（4）什么是安全性测试？它与可靠性测试有什么区别？
（5）Web 测试策略是什么？



面向对象测试

本章介绍面向对象测试，就面向对象分析测试、面向对象设计测试和面向对象编程测试分别给出详细说明，并对面向对象的单元测试、集成测试和系统测试给出解释。

7.1 面向对象影响测试

对象、类作为构成面向对象程序的基本元素，封装了数据及作用在数据上的操作，父类中定义共享的公共特征，子类继承父类所有特征外，并引入新的特征，使得类和类之间通过继承形成有向无圈图结构。

传统软件的测试往往关注模块的算法细节和模块接口间流动的数据，面向对象软件的类测试由封装在类中的操作和类的状态行为所驱动。封装、继承、多态等面向对象的特性一方面提高了软件开发效率，保证了软件都质量，但另一方面也给软件测试提出了新的问题，增加了测试的难度和复杂性。

下面具体分析面向对象对软件测试的影响。

1. 封装性影响测试

类的重要特征之一是信息隐蔽，其通过对象的封装性实现。封装是将一个对象的各个部分聚集在一个逻辑单元内，对象的访问被限制在接口上，限制了外界对于对象属性的可视性与操作行，减低类和程序其他各部分之间的依赖，使得程序模块化，避免外界对类不合理操作，从而防止错误的扩散。但是，信息隐蔽却给测试带来许多问题。面向对象软件中，对象行为是被动的，在接收到相关外部信息后才被激活，对象的状态可能发生变化而进入新的状态。由于信息隐蔽与封装机制，类的内部属性和状态对外界是不可见的，只能通过类自身的方法获得，这给类测试时测试用例执行是否处于预期状态的判断带来困难。

2. 继承性影响测试

在面向对象程序中，继承由扩展、覆盖和特例化三种基本机制实现。其中扩展是子类包含父类的特征；覆盖是子类的方法与父类的方法有相同的名字和消息参数，但其实现的方法不同；特例化是子类中特有的方法和实例变量。继承有利于代码的复用，但同时也使错误传播概率提高。

Weyuker 提出基于继承测试数据集的充分性公理，如下所示。

1）反扩展性公理

反扩展性公理认为若有两个功能相同而实现不同的程序，对其中一个是充分的测试数据集未必对另一个是充分的测试数据集。这一公理表明在子类中重定义了某一继承的方法，即使两个函数完成相同的功能，对被继承方法是充分的测试未必对重定义的方法是充分的。

2）反分解性公理

反分解性公理认为一个程序进行过充分的测试，并不表示其中的成分都得到了充分的测试。因为这些独立成分有可能被用在新的环境中，需要在新的环境中对其重新进行测试。因此，若一个类得到了充分的测试，当其被子类继承后，继承的方法在子类的环境中的行为特征需要重新测试。

3）反组合性公理

反组合性公理认为一个测试即使对于程序中各个单元都是充分的，也并不表示对整个程序是充分的，因为独立部分交互时会产生在隔离状态下所不具备的新特性。这一公理表明，若对父类中某一方法进行了重定义，仅对该方法自身或其所在的类进行重新测试是不够的，还必须测试其他有关的类（如子类和引用类）。

Perry 和 Kaiser 对 Weyuker 观点总结如下：随着继承层次的加深，可供重用的类越来越多，编程效率也越来越高，但无形中加大了测试的工作量和难度。同时，递增式软件开发过程中，如果父类发生修改，这种变化会自动传播到所有子类，使得父类、子类都必须重新测试。所以说，继承性使得测试更加复杂。

3. 多态性影响测试

多态性是将多种不同的特殊行为进行抽象的一种能力，对于同样的消息被不同类型的对象接收时导致完全不同的行为，使得面向对象程序对外呈现出强大的处理能力，但同时却使得程序内“同一”函数的行为复杂化，多态促成了子类型替换。一方面，子类型替换使对象的状态难以确定。如果一个对象包含了 A 类型的对象变量，则 A 类型的所有子类型的对象也允许赋给该变量。程序运行过程中，该变量可能引用不同类型的对象，其结构不断变化。另一方面，子类型替换使得向父类对象发送的消息也允许向子类对象发送。

由此可见，多态性和动态绑定使得系统能自动为给定消息选择合适的实现代码，可是，由于其的不确定性，增加测试用例的选取难度。

7.2 面向对象测试模型

面向对象开发模型分为面向对象分析、面向对象设计和面向对象编程三个阶段。面向对象分析阶段产生整个问题空间的抽象描述，从系统能完成的功能，以及对象间的相互关联关系为核心，分析主要围绕对象的分类，各个相关属性和操作的标识，类和实例之间

的关系，各个对象的行为特征等方面进行。

面向对象设计描述软件如何才能满足需求。面向对象设计是分析的进一步细化和扩充，重点在于说明项目的实施方案确定类和类结构。设计不仅要满足当前需求分析的要求，更重要的是要能方便地实现功能的重用和扩充，不断地适应用户的要求。

面向对象编程是选择适合于面向对象编程语言的类和类结构，形成代码，是将面向对象设计模型用编程语言进行描述。

面向对象开发模型中，分析模型映射为设计模型，设计模型又映射为源程序代码，具有一致性和统一性。面向对象测试模型能有效地将分析、设计的文本或图表代码化，测试模型如图 7.1 所示。

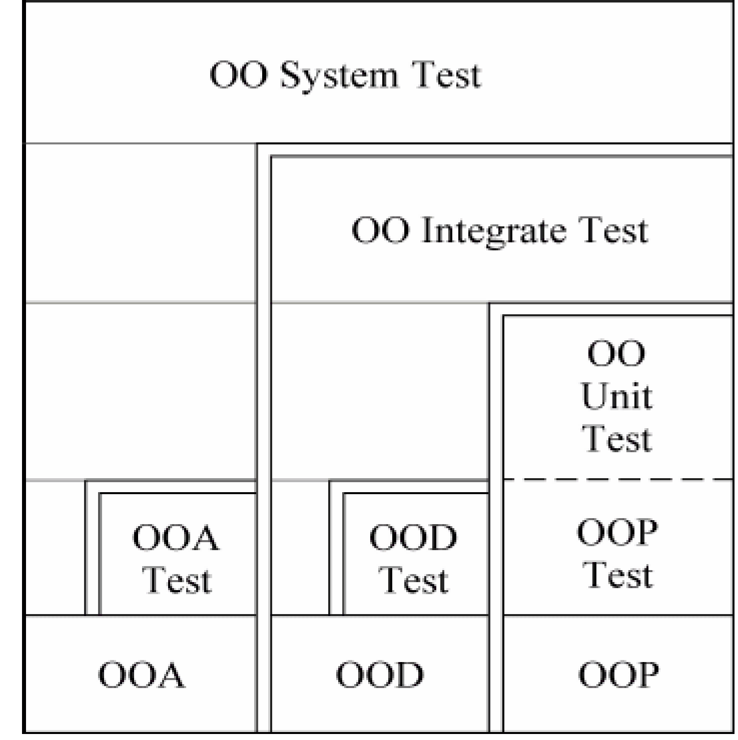


图 7.1　面向对象测试模型

其中，OOA Test 是面向对象分析测试；OOD Test 指面向对象设计测试，OOA Test 和 OOD Test 主要对分析设计文档进行，是软件开发前期的关键性测试。OOP Test 是面向对象编程测试，主要针对编程风格和程序代码进行测试。OO Unit Test 是指面向对象单元测试，对程序单元的功能模块测试；OO Integrate Test 是指面向对象集成测试，主要对系统单元模块之间的相互服务进行测试，如成员函数间的相互作用，类之间的消息传递等；OO System Test 是指面向对象系统测试，主要以系统的需求规格说明为测试标准。

7.3　面向对象分析测试

面向过程分析是功能分解的过程，着眼点在于一个系统需要什么样的信息处理方法和过程。面向对象分析直接映射需求分析问题，将问题空间功能抽象化，用对象的结构反映实例和实例之间的复杂关系，面向对象分析为类的实现以及类层次结构的组织和实现提供平台。

面向对象分析测试分为五个方面：对认定的对象的测试、对认定的结构的测试、对认定的主题的测试、对定义的属性和实例关联的测试、对定义的服务和消息关联的测试。

7.3.1　对象测试

OOA 中认定的对象是对问题空间中实例的抽象，从以下方面对其进行测试：

(1) 认定的对象是否全面，问题空间中所有的实例是否都反映在认定的抽象对象中。

(2) 认定的对象是否具有多个属性。将唯一一个属性的对象作为其他对象的属性处理，而不是抽象为独立的对象。

(3) 对认定为同一对象的实例是否有共同的，区别于其他实例的共同属性。

(4) 对认定为同一对象的实例是否提供相同的服务，如果服务随着不同的实例而变化，认定的对象就需要分解或利用继承性来分类表示。

(5) 系统没有必要始终保持对象代表的实例信息，提供或者得到关于它的服务，认定的对象也无必要。

(6) 认定的对象的名称应该尽量准确、适用。

7.3.2 结构测试

在面向对象的 Coad 方法中，Coad 分析共有 5 个层次，分别是发现类及对象、定义属性、定义服务、识别结构和定义主题。其中，结构是多种对象的组织方式，用来反映问题空间中的复杂实例和复杂关系。

在 Coad 方法中，结构是多种对象的组织方式，用来反映问题空间中的复杂实例和复杂关系。认定的结构分为两种：分类结构和组装结构。其中，分类结构体现了问题空间中实例的一般与特殊关系，组装结构体现了问题空间中实例整体与局部的关系。

1. 对认定的分类结构的测试

(1) 对于结构中处于高层的对象，是否在问题空间中含有不同于下一层对象的特殊可能性，即是否能派生出下一层对象。

(2) 对于结构中处于同低层的对象，是否能抽象出在现实中有意义的更一般的上层对象。

(3) 对所有认定的对象，是否能在问题空间内抽象出在现实中有意义的对象。

(4) 高层的对象的特性是否完全体现下层的共性。

(5) 低层的对象是否具有高层对象特性基础上的特殊性。

2. 对认定的组装结构的测试

(1) 整体和部件的组装关系是否符合现实的关系。

(2) 整体和部件是否在考虑的问题空间中有实际应用。

(3) 整体中是否遗漏了反映在问题空间中有用的部件。

(4) 部件是否能够在问题空间中组装新的有现实意义的整体。

7.3.3 主题测试

主题如同文章中内容的概要，是在对象和结构基础上抽象，其提供 OOA 分析结果的可见性。对主题层的测试应该考虑以下方面：

(1) 贯彻 George Miller 的“7+2”原则，如果主题个数超过 7 个，就要对相关密切的主题进行归并。

(2) 主题所反映的一组对象和结构是否具有相同和相近的属性和服务。

(3) 认定的主题是否是对象和结构更高层的抽象，是否便于理解 OOA。

(4) 主题间的消息联系是否代表了主题所反映的对象和结构之间的所有关联。

7.3.4　属性和实例关联测试

属性用来描述对象或结构所反映的实例特性。实例关联是反映实例集合间的映射关系。对属性和实例关联的测试从如下方面考虑：

(1) 定义的属性是否对相应的对象和分类结构的每个实例都适用。

(2) 定义的属性在现实世界是否与这种实例关系密切。

(3) 定义的属性在问题空间是否与这种实例关系密切。

(4) 定义的属性是否能够不依赖于其他属性被独立理解。

(5) 定义的属性在分类结构中的位置是否恰当，低层对象的共有属性是否在上层对象属性体现。

(6) 在问题空间中每个对象的属性是否定义完整。

(7) 定义的实例关联是否符合现实。

(8) 在问题空间中实例关联是否定义完整，特别需要注意一对多和多对多的实例关联。

7.3.5　服务和消息关联测试

服务定义了每种对象和结构在问题空间所要求的行为。问题空间中实例的通信在OOA 中相应地定义为消息关联。对定义的服务和消息关联的测试从如下方面进行：

(1) 对象和结构在问题空间的不同状态是否定义了相应的服务。

(2) 对象或结构所需要的服务是否都定义了相应的消息关联。

(3) 定义的消息关联所指引的服务提供是否正确。

(4) 沿着消息关联执行的线程是否合理，是否符合现实过程。

(5) 定义的服务是否重复，是否定义了能够得到的服务。

面向对象分析的测试如表 7.1 所示。

表 7.1　面向对象分析的测试

测试内容	概　　述	测试考虑方面
认定对象的测试	认定的对象：对问题空间中的结构、其他系统、设备、被记忆的事件、系统涉及的人员等实际的抽象	(1) 是否全面，问题空间中的实例是否都反映在认定的抽象对象中 (2) 是否具有多个属性，只具有一个属性的对象不抽象为独立的对象 (3) 对认定为同一对象的实例是否有共同的、区别于其他实例的共同属性 (4) 对认定为同一对象的实例是否提供或需要相同的服务，如果服务随着实例的不同而变化，认定的对象就需要分析或利用继承性来分类表示 (5) 如果系统没有必要始终保持对象代表的实例信息，提供或者得到关于它的服务、认定的对象也无必要 (6) 认定的对象的名称要尽量准确、适用

续表

测试内容	概　　述		测试考虑方面
认定结构的测试	认定结构：多种对象的组织方式，用来反映问题空间中的复杂实例和复杂关系，认定的结构分为两种：分类结构和组装结构	分类结构：体现问题空间中实例的一般与特殊的关系	(1) 结构中一种对象尤其是高层对象，是否存在不同于下一层对象的特殊的可能性，即是否能派生出下一层对象 (2) 结构中一种对象尤其是同一低层对象，是否能抽象出在现实中有意义的更一般的上层对象 (3) 对所有认定的对象，是否能向上层抽象出现实中有意义的对象 (4) 高层的对象的特性是否完全体现下层的共性 (5) 低层的对象是否有高层特性基础上的特殊性
		组装结构：体现问题空间中实例的整体与局部的关系	(1) 整体对象和部件对象的组装关系是否符合现实的关系 (2) 整体对象和部件对象是否在考虑的问题空间中有实际关系 (3) 整体对象是否遗漏了在问题空间中有用的部件对象 (4) 部件对象是否能够在问题空间中组装成新的有意义的整体对象
认定主题的测试	主题：在对象和结构的基础上更高一层的抽象，是为了提供 OOAOOA 分析结果的可见性，如同文章中各章的摘要		(1) 贯彻 George Mille：的“7＋2”原则。如果主题个数 A 超过 7 个，归并有较密切属性和服务的主题 (2) 主题所反映的一组对象和结构是否具有相同或相近的属性和服务 (3) 认定的主题是否是对象和结构更高一层的抽象，是否便于理解 OOA 结果的概括 (4) 主题间的消息联系(抽象)是否代表主题所反映的对象和结构之间的所有关联
对定义的属性和实例关联的测试	属性：用来描述对象或结构所反映的实例的特性；实例关联：反映实例集合间的映射关系		(1) 定义的属性是否对相应的对象和分类结构的每个现实实例都适用 (2) 定义的属性在现实世界是否与这种实例关系密切 (3) 定义的属性在问题空间是否与这种实例关系密切 (4) 定义的属性是否能够不依赖于其他属性被独立理解 (5) 定义的属性在分类结构中的位置是否恰当，低层对象的共有属性是否在上层对象属性中体现 (6) 在问题空间中每个对象的属性是否定义完整 (7) 定义的实例关联是否符合现实 (8) 在问题空间中实例关联是否定义完整，特别需要注意“一对多”、“多对多”的实例关联
对定义的服务和消息关联的测试	定义的服务：定义的每一种对象和结构在问题空间所要求的行为；消息关联：题空间中实例之间必要的通信，需要定义相应的消息关联		(1) 对象和结构在问题空间中的不同状态是否定义相应的服务 (2) 对象和结构所需的服务是否都定义了相应的消息关联 (3) 定义的消息关联所指引的服务是否正确 (4) 沿着消息关联执行的线程是否合理，是否符合现实过程 (5) 定义的服务是否重复，是否定义了能够得到的服务

7.4　面向对象设计测试

结构化设计方法采用面向作业的设计方法，把系统分解为一组作业。面向对象设计采用“造型的观点”，是以 OOA 为基础归纳出类，建立类结构，实现分析结果对问题空间的抽象，设计类的服务。由此可见，OOD 是 OOA 的进一步细化和抽象，其界限通常难以

严格区分。OOD确定类和类结构不仅是满足当前需求分析的要求,更重要的是通过重新组合或加以适当的补充,实现功能的重用和扩增。

因此,OOD测试从对认定的类的测试、对构造的类层次结构的测试和对类库的支持的测试三方面考虑。

1. 对认定类测试

OOD认定的类是OOA中认定的对象,是对象服务和属性的抽象。认定的类应该尽量是基础类,这样便于维护和重用。

测试认定的类有一些准则:

(1) 是否涵盖了OOA中所有认定的对象。

(2) 是否能体现OOA中定义的属性。

(3) 是否能实现OOA中定义的服务。

(4) 是否对应一个含义明确的数据抽象。

(5) 是否尽可能少地依赖其他类。

2. 对类层次结构测试

OOD的类层次结构基于OOA的分类结构产生,体现了父类和子类之间的一般性和特殊性。类层次结构是在解空间构造实现全部功能的结构框架。测试如下方面:

(1) 类层次结构是否涵盖了所有定义的类。

(2) 是否能体现OOA中所定义的实例关联。

(3) 是否能实现OOA中所定义的消息关联。

(4) 子类是否具有父类没有的新特性。

(5) 子类间的共同特性是否完全在父类中得以体现。

3. 对类库支持测试

类库主要用于支持软件开发的重用,对类库的支持属于类层次结构的组织问题。由于类库并不直接影响软件的开发和功能实现,因此,类库的测试往往作为对高质量类层次结构的评估。其测试点如下:

(1) 一组子类中关于某种含义相同或基本相同的操作,是否有相同的接口。

(2) 类中方法功能是否较单纯,相应的代码行是否较少,一般建议为不超过30行。

(3) 类的层次结构是否是深度大、宽度小。

7.5 面向对象单元测试

面向对象软件测试过程以层次增量的方式进行。首先对类的函数进行测试;然后,对类进行测试;再次,将多个类集成为类簇或子系统进行集成测试;最后,进行系统测试。其中,面向对象单元测试针对类中的成员函数以及成员函数间的交互进行测试;面向对象集成测试主要对系统内部的相互服务进行测试,如类之间的消息传递等;面向对象系统测试

是基于面向对象集成测试的最后阶段的测试，主要以用户需求为测试标准。

下面，介绍类的测试方法。

7.5.1 功能性和结构性测试

类测试有两种主要的方式：功能性测试和结构性测试。功能性测试和结构性测试分别对应传统测试的黑盒测试和白盒测试。测试类的方法，对方法调用关系进行测试。测试每个方法的所有输入情况，并对这些方法之间的接口进行测试。对类的构造函数参数以及消息序列进行选择保证其在状态集合下正常工作。因此，对类的测试分成如下两个层次：方法内测试和方法间测试。

1. 方法内测试

方法内测试作为第一个层次，考虑类中单独方法，这个层次的测试等效于传统程序中单个过程的测试，传统测试技术，如黑盒测试方法：等价类划分、边界值分析和错误推测等以及白盒测试方法：逻辑覆盖方法仍是测试类中每个方法的主要手段。

面向对象软件中方法的执行是通过消息驱动执行的。测试类中的方法，必须用驱动模块对被测方法通过发送消息来驱动执行，被测试模块或者方法调用其他模块或方法，则需要设计一个模拟被调程序功能的桩模块。驱动模块、桩模块及被测模块组成测试环境。

2. 方法间测试

方法间测试作为第二个层次，考虑类中方法之间的相互作用，对方法进行综合测试。单独测试一个方法时，只考虑其本身执行的情况，而没有考虑方法的协作关系。方法间测试考虑一个方法调用本类中的其他方法，或其他类的方法。

类的操作被封装在类中，对象之间通过发送消息启动操作，对象作为一个多入口模块，必须考虑测试方法的不同次序组合的情况，当一个类中方法的数目较多时，次序的组合数目将非常多。对于操作的次序组合以及动作的顺序问题，测试用例中加入了激发调用信息，检查它们是否正确运行。对于同一类中方法之间的调用，遍历类的所有主要状态。同时，选出最可能发现属性和操作错误的情况，重点进行测试。

7.5.2 测试用例设计和选择

传统面向过程的软件测试用例设计从软件的各个模块算法出发，而面向对象软件测试用例设计对面向对象的特性，如封装性、继承性和多态性等进行测试，Berard 提出测试用例的设计方法，关于设计合适的操作序列以测试类的状态，主要原则包括：

（1）对每个测试用例应当给予特殊的标识，并且还应当与测试的类有明确的联系。

（2）测试目的应当明确。

（3）应当为每个测试用例开发一个测试步骤列表，列表包含以下内容：

① 列出所要测试对象的说明。

② 列出将要作为测试结果的消息和操作。

③ 列出测试对象可能发生的例外情况。

④ 列出外部条件，为了正确对软件进行测试所必须有的外部环境的变化。

⑤ 列出为了帮助理解和实现测试所需要的附加信息。

为了合理地控制测试用例的数目，往往采用其于概率分布的测试用例抽样的方法。其中，总体是指所有可能被执行的测试用例，包括所有前置条件和所有输入值可能的组合情况。样本是基于概论分布选择的子集，子集的使用频率越高，被选中的概率越大。样本集合中每个样本代表一个特定的个体。例如，用例模型作为测试用例分层的基础，挑选出一个测试用例的抽样，选择一个测试系列，并不要求一定要首先明确如何来确定测试用例的总体。构建测试用例的一个测试系列，将类说明作为测试用例的来源，运用一种抽样方法对测试进行补充，减少测试的数目。

7.6　面向对象集成测试

传统面向过程的软件模块具有层次性，模块之间存在着控制关系。而面向对象的软件，其功能散布在不同类中，通过消息传递提供服务，没有传统面向过程的软件模块的层次控制结构，其通过构成类的各个部件之间存在直接和非直接交互，软件的控制流无法确定，从而传统软件自顶向下和自底向上的组装策略意义不大，采用传统的将操作组装到类中的增殖式组装常常行不通。

集成测试关注于系统的结构和类之间的相互作用，测试步骤一般首先进行静态测试，然后进行动态测试。静态测试主要针对程序的结构进行，检测程序结构是否符合设计要求，采用逆向工程测试工具得到类的关系图和函数关系图，与面向对象设计规格说明比较检测程序结构和实现上是否有缺陷，是否符合需求设计。动态测试根据功能结构图、类关系图或者实体关系图，确定不需要被重复测试的部分，通过覆盖标准减少测试工作量。

通过下列步骤设计测试用例：

(1) 选定检测的类，参考 OOD 分析结果，得到类的状态和行为，类或成员函数间传递的消息，输入或输出的界定等数据。

(2) 确定采用什么样的覆盖标准。

(3) 利用结构关系图确定待测类的所有关联。

(4) 根据程序中类的对象构造测试用例，确认使用什么输入激发类的状态、使用类的服务和期望产生什么行为等。

面向对象软件由若干对象之间的相互协作实现功能。相互协作是一系列参与交互的对象协作中的消息的集合。例如，对象作为参数传递给另一对象时，或者当一个对象包含另一对象的引用并将其作为这个对象状态的一部分时，对象的交互就会发生。

对象交互的方式有如下几类：

(1) 公共操作将一个或多个类命名为正式参数的类型。

(2) 公共操作将一个或多个类命名为返回值的类型。

(3) 类的方法创建另一个类的实例，并通过该实例的调用操作。

(4) 类的方法引用某个类的全局实例。

交互测试的重点是确保对象之间进行消息传递,由于可能发生多重的对象交互,需要考虑交互对象内部状态的影响,以及相关对象的影响。这些影响主要包括:涉及对象的部分属性值变化,涉及对象的状态变化,创建新对象和删除对象而发生的变化。

进行交互测试时,具有以下几个特点:

(1) 假定相互关联的类都已经被充分测试。

(2) 交互测试建立在公共操作上,相对于建立在类实现的基础上要简单。

(3) 采用一种公共接口方法,将交互测试限制在与之相关联的对象上。

(4) 根据每个操作说明选择测试用例,并且这些操作说明都基于类的公共接口上。

面向对象软件中类分为原始类和非原始类。原始类是最简单的组件,其数目较少。非原始类是指在某些操作中支持或需要使用其他对象的类。根据非原始类与其他实例交互的程度,非原始类分为汇集类和协作类。下面具体介绍汇集类和协作类测试。

(1) 汇集类是指有些类的说明中使用对象,但是实际上从不和这些对象进行协作。编译器和开发环境的类库通常包含汇集类。例如,C++ 的模板库、列表、堆栈、队列和映射等管理对象。汇集类一般具有如下行为:

① 存放这些对象的引用。

② 创建这些对象的实例。

③ 删除这些对象的实例。

(2) 凡不是汇集类的非原始类就是协作类。协作类是指在一个或多个操作中使用其他的对象并将其作为实现中不可缺少的一部分。协作类测试的复杂性远远高于汇集类的测试,协作类测试必须在参与交互的类的环境中进行测试,需要创建对象之间交互的环境。

系统交互既发生在类内的方法之间,也发生在不同的类之间。类 A 与类 B 交互往往有如下两种情况。

① 类 B 的实例变量作为参数传给类 A 的某方法,类 B 的改变必然导致对类 A 的方法的回归测试。

② 类 A 的实例作为类 B 的一部分,类 B 对类 A 中变量的引用需进行回归测试。

由于交互测试的粒度与缺陷的定位密切相关,粒度越小越容易准确定位缺陷。但是,粒度小使得测试用例数和测试执行开销增加。因此,必须权衡测试所需资源和测试粒度之间的关系,选择合理的交互测试所需的测试粒度。

被测交互聚合块大小的选择,需要考虑以下三个因素:

(1) 区分那些与被测对象有组成关系的对象和那些仅仅与被测对象有关联的对象。在类测试期间,测试组合对象与其组成属性之间的交互。集成测试时,测试对象之间的交互。

(2) 交互测试期间所创建的聚合层数与缺陷的能见度紧密相关,若“块”太大,会有不正确的中间结果。

(3) 对象关系越复杂,一轮测试之前被集成的对象应该越少。

7.7 面向对象系统测试

单元测试和集成测试仅能保证软件开发的功能的实现，但不能确保在软件在实际运行中是否满足用户的需要，因此，必须对软件进行规范的系统测试。系统测试着眼于用户的需求，测试软件与系统其他部分配套运行情况，保证系统各部分在协调的环境下正常工作。

系统测试参照面向对象分析模型，测试组件序列中的对象、属性和服务。组件是由若干类构建，首先实施接受测试。接受测试将组件放在应用环境中，检查类的说明，采用极值甚至不正确数值进行测试。其次，组件的后续测试顺着主类的线索进行。

7.8 习　　题

（1）什么是汇集类？什么是协作类？怎样测试汇集类和协作类？

（2）类测试的方法有哪些？类测试分几个层次？

（3）对 OOA 阶段的测试划分为几个方面？分别是什么？

（4）软件测试模型是什么？

（5）测试抽象类有哪些方法？各自的优缺点是什么？



嵌入式软件测试

本章首先介绍了嵌入式系统的概念、软件架构和开发方式;其次,对嵌入式软件测试的特点、测试策略、测试流程和三种测试环境进行了说明;最后,介绍 3 类嵌入式软件测试工具。

8.1 嵌入式系统

8.1.1 基本概念

IEEE 认为嵌入式系统(Embedded System)是“用于控制、监视或者辅助操作机器和设备的装置”。当前国内普遍认同的嵌入式系统定义为:“以应用为中心和以计算机技术为基础的,并且软硬件可裁减的,能满足由于系统对功能、可靠性、成本、体积、功耗等指标的严格要求的专用计算机系统”。嵌入式系统通常由嵌入式处理器、嵌入式外围设备、嵌入式操作系统以及用户的应用程序等四个部分组成。与通用计算机系统相比具有以下特点:

(1) 实时性:实时性是指软件必须在可预知的时间内得出正确的执行结果,否则视为无效。嵌入式系统对于时间特性要求非常严格,无论系统是否处于峰值状态,对紧急事件必须在确定的时限内给出响应,从而使系统具有可预测性。

(2) 嵌入性:嵌入性是指系统 A 内置到另一个更大的系统 B 中,则称为 A 嵌入 B。一般地,嵌入式系统在更大的系统中提供控制和计算功能,用于管理和控制。

(3) 反应性:反应性是指与外部环境交互的反应性是指根据外部事件做出响应的特性。嵌入式系统由事件驱动,必须对外界事件做出响应即需要从外部接收数据,做出控制决策,然后提供输出并控制外部环境。

(4) 专用性:嵌入式系统软件往往只能在特定的专用系统中工作,无法运行于其他系统中。

(5) 体积小:嵌入式系统通常嵌入到相关硬件中,由封装好的软件系统控制相关硬件,处于对产品体积、成本等因素的考虑,要求所占用的空间尽可能地小,同时要求应用程序占用的内存较小。

(6) 依赖性:嵌入式系统软件与硬件有紧密的联系,对硬件具有很强的依赖性。

8.1.2 嵌入式系统软件架构

和普通计算机系统一样，嵌入式系统除了需要硬件结构和配置外，还需要相应软件的支持才能完整地实现系统的功能。目前，嵌入式系统的软件体系结构通常都采用以实时内核为基础的分层体系结构。图 8.1 给出了嵌入式系统软件分层图。

应用层
中间层
操作系统层
驱动层
硬件

图 8.1 嵌入式系统软件结构

(1) 驱动层：驱动层又名硬件抽象层，是最靠近硬件的一层软件，直接和硬件打交道，为操作系统和应用提供使用硬件的接口即驱动的支持。

(2) 操作系统层：又名实时内核层，主要功能是协作最顶层的应用层更好地进行任务的调度、消息管理和异常处理等工作。当前流行的商用嵌入式操作系统有 Win-driver 公司的 VxworkS，微软公司的 Windows CE，Sun 公司的 Java OS，国内的 Delta OS 等；开放源码的嵌入式操作系统有各种嵌入式 Linux、eCos、μC/OS 等。

(3) 中间层：又称为应用软件实现支持层。目前，嵌入式软件编程语言主要有面向过程编程语言和面向对象编程语言等，必须通过相应的编译器或解释器的支持，才能在嵌入式系统中运行。

(4) 应用层：应用层软件由多个相对独立的应用任务组成，如 I/O 任务、计算任务和通信任务等，各个任务的运行由操作系统进行调度。

8.1.3 嵌入式系统开发方式

嵌入式系统通常只为软件提供执行环境（运行环境），而不提供软件的开发环境（宿主机环境）。因此，嵌入式系统是一个资源受限的系统，决定了嵌入式软件开发必须有一套专门的开发环境。

嵌入式软件的开发环境和运行环境往往互相分离，采用交叉开发的方式，即编辑和编译软件在宿主机上运行，编译的可执行软件在目标机上运行，如图 8.2 所示。

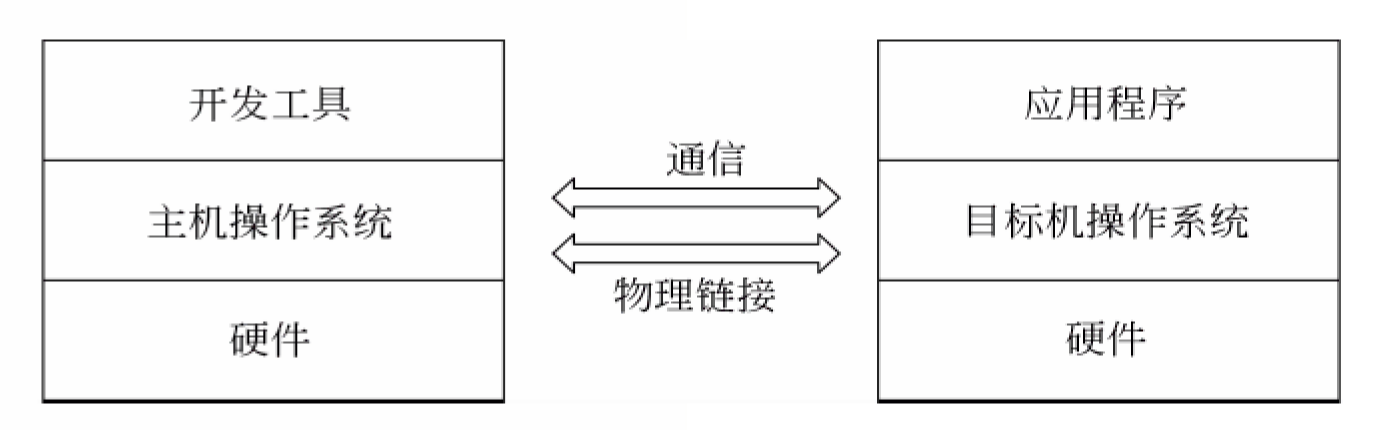


图 8.2 嵌入式交叉开发方式

宿主机(Host)是一台通用计算机，一般是 PC 和工作站，通过串口或网络连接与目标机通信。目标机(Target)是嵌入式系统软件运行的硬件平台，其硬件资源有限，在目标机上运行的软件可剪裁。

嵌入式交叉开发需要交叉开发工具，包括交叉编译器、交叉调试器和系统仿真器等。其中交叉编译器用于在宿主机上生成能在目标机上运行的代码，交叉调试器和系统仿真

器则用于在宿主机与目标机间完成程序代码的调试。

8.2 嵌入式软件测试

8.2.1 测试特点

由于嵌入式软件的开发环境和运行环境的不一致，嵌入式软件测试面临着目标环境和宿主环境的选择测试问题，嵌入式软件的实时性、嵌入性和反应性等特性对测试具有一定影响，下面依次进行介绍。

1. 实时性对测试的影响

时间特性作为嵌入式软件测试的核心问题之一，对其时间特性测试分为两种方法：即静态时间分析和动态实时检测。

1）静态时间分析

静态时间分析就是不执行被测程序，而通过分析程序结构来预估程序执行时间的方法。由于不能准确估计分支和循环等因素所占时间，因此不能确定程序的实际执行时间。但静态分析可确定程序在最坏情况下的执行时间，即程序最大执行时间是否满足时间约束。

2）动态实时检测

动态实时检测就是通过执行程序来测试程序的时间特性。在线仿真器 ICE、指令仿真器和插桩工具是三种最常用的方法。

2. 嵌入性对测试的影响

嵌入式软件测试的一个重要问题是建立宿主机与目标机之间的物理/逻辑连接，解决数据信息的传输问题。由于嵌入式软件运行在目标机上，因此，即使在宿主机环境下测试再充分，也不能说明在目标机环境下该软件运行不出问题。因而，嵌入式软件必须对目标环境进行测试。

3. 反应性对测试的影响

反应式系统在任何时刻都要对可能出现的事件作出适当反应。“激励-响应”反应式系统输入事件各种序列的组合给测试工作带来困难，如何选取测试用例成为反应式软件的关键问题之一。

8.2.2 测试策略

在嵌入式软件测试中，常常要在基于目标机的测试和基于宿主机的测试之间进行折中。一般在宿主机中，进行与硬件无关的逻辑或界面测试。而代码测试、中断测试和硬件接口测试往往只能在目标环境中进行。

嵌入式软件测试策略主要有交叉调试、目标代理等方式。

1. 交叉调试

对于主机和目标机之间的通信连接,可以通过串口通信方式,也可以是以太网口,一般基于 TCP/IP 协议传输。在嵌入式操作系统中,宿主机和目标机处于不同的机器中,宿主机要对目标机程序进行调试及测试控制,捕捉目标机上被测试程序是否正常接受测试数据,可以通过交叉调试的方式来实现,如图 8.3 所示。

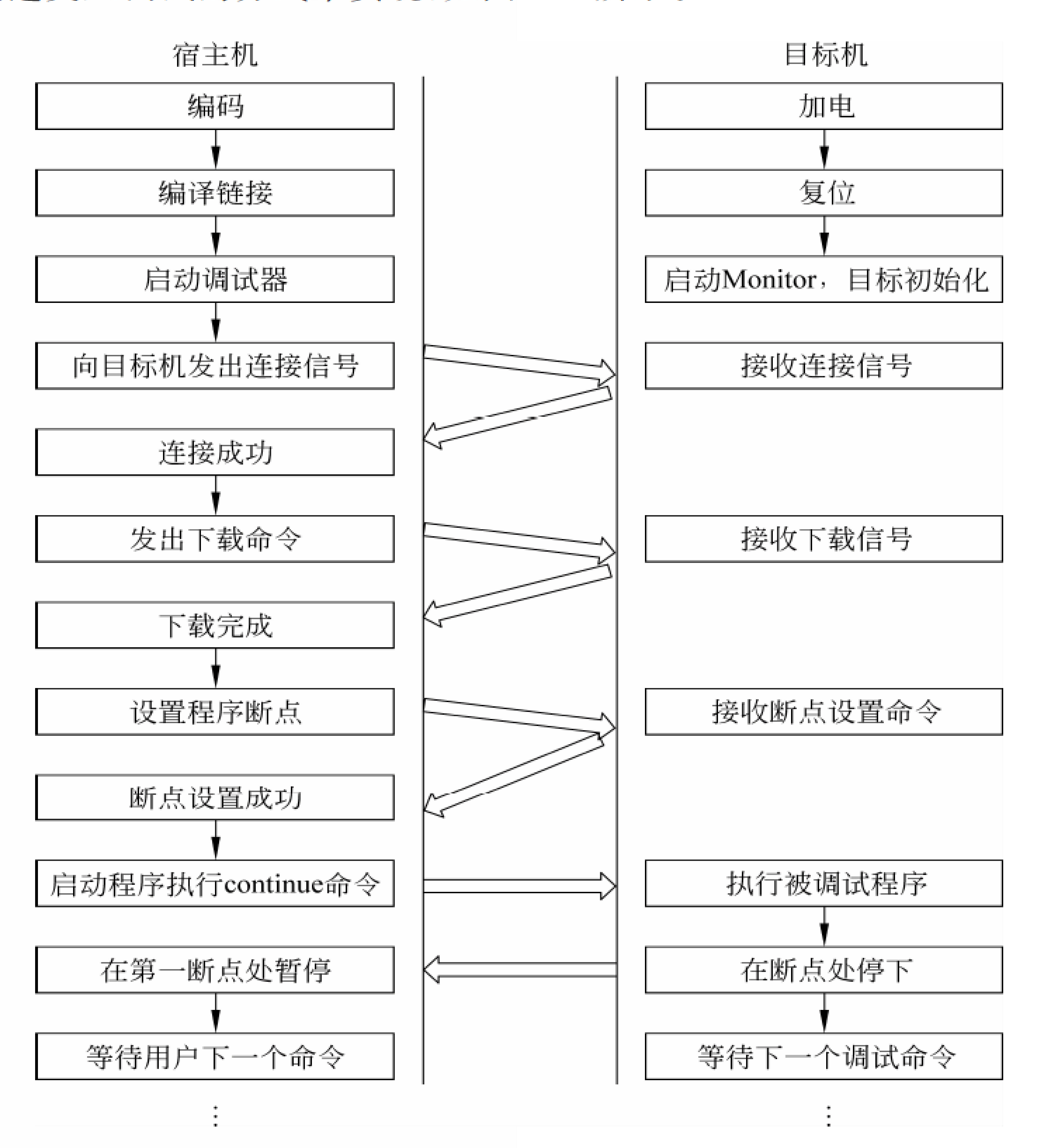


图 8.3　交叉调试方式

交叉调试也称远程调试,即调试器运行于宿主机的桌面操作系统上,而被调试/测试程序运行于目标机的嵌入式操作系统上。交叉调试允许调试器以某种方式控制被调试进程的运行方式,并具有查看和修改目标机上内存单元、寄存器以及被调试进程中变量值等各种调试功能。

2. 目标代理

对于目标机如何反馈测试信息及测试信息在宿主机端的显示。目标机的被测试程序在获取输入后,进行相应处理,产生正常响应信息或抛出异常、错误信息。这些信息的反

馈由于要受到目标机资源的限制，一般都是通过宿主机和目标机之间的通信连接，返回给宿主机处理。由于目标操作系统的所有异常处理最终都必须转向通信模块，通知调试器异常，调试器依据该异常向用户显示被调试/测试程序发生了哪一类异常现象。为此在目标系统中引入控制功能的目标代理模块——监视器。

监视器负责与调试器共同配合以完成对目标机上运行的进程进行调试及测试，这种方式构建交叉测试方式如图8.4所示。

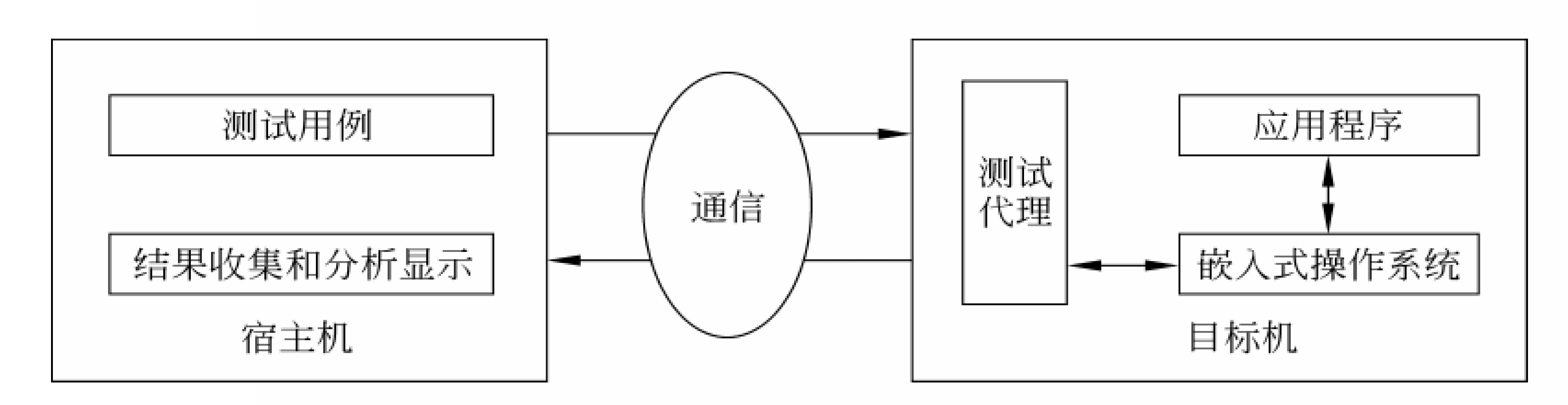


图8.4　交叉调试方式

目标机上的目标代理要具备以下功能：

(1) 具备通信服务程序的功能。可以通过串口或以太网口接收主机输入，并能将信息回传给主机的调试器。

(2) 具备"中间件"的功能。对于外部主机而言，它是与其直接接触的"目标机代理"；对于目标机上的被测试软件功能来说，它是外部测试用例输入的"测试代理"。

(3) 具有中间件功能，可以控制测试数据流在目标机的被测试程序上的输入和输出，并将其反馈给主机处理。

8.2.3　三种测试环境

实践中，嵌入式软件的专用性决定了其测试需要专用的测试环境，一般有宿主机软件仿真、在线仿真器ICE和目标机仿真三种测试环境。

1. 宿主机软件仿真

宿主机软件仿真是用软件构造一个嵌入式应用程序运行所需的仿真环境，能模拟执行目标机CPU的指令，还能模拟中断、I/O命令等外部消息。其优点是无须目标机硬件便可测试，灵活、方便；自动化程度较高；开发成本较低。但由于实时性受限，无法模拟真实硬设备之间的高速通信。因此，宿主机软件仿真适用于实时性要求不高的嵌入式系统。

2. 在线仿真器ICE

在线仿真器ICE是仿照目标机上的CPU而专门设计的硬件，可以完全仿真处理器芯片的行为，并且提供了非常丰富的调试功能。它配有专用于特定CPU芯片的接头，能提供与应用程序交互的软信道或硬信道。在线仿真器能发现现实世界中的各种信息，因此尽管在线仿真器的价格非常昂贵，仍然得到了广泛的应用。其不足是硬件依赖性强，测试范围受限。

3. 目标机仿真

目标机仿真提供应用程序实际的运行环境，测试结果真实，但要受到目标机的硬件限制，难以区分软件和硬件的错误。

在实际测试工作中，应综合考虑成本、测试类型、测试可靠性以及项目进度等因素选择测试环境。一般而言，ICE 较多用于程序开发和调式/测试阶段，测试可先在宿主机上进行，再在目标机环境下确认测试。通常在宿主机环境执行多数的测试，只是在最终确定测试结果和最后的系统测试才移植到目标环境。

8.2.4 测试流程

嵌入式软件测试经历单元测试、集成测试、系统测试、硬件软件集成测试等四个阶段，前三个阶段适用于任何软件的测试，硬件软件集成测试阶段作为嵌入式软件特有，其目的是验证嵌入式软件与其他软硬件设备是否正确地交互。

下面依次介绍嵌入式软件的四个测试阶段。

1. 单元测试

嵌入式测试系统单元测试包括程序的插桩过程（包括预处理和词法语法分析）、测试用例的生成、动态测试信息的分析。被测试程序首先经过预处理，主要是进行宏替换和将短跳转改为远跳转；然后进行语法词法分析，对整个程序进行扫描后，生成一些相互关联的链表，用与确定插桩函数的位置；在关键的字段和函数处插桩，选择逻辑测试类型，编译生成含有插桩函数的目标文件，记录该测试类型中的所有函数的位置，自动生成相应的测试用例，如图 8.5 所示。

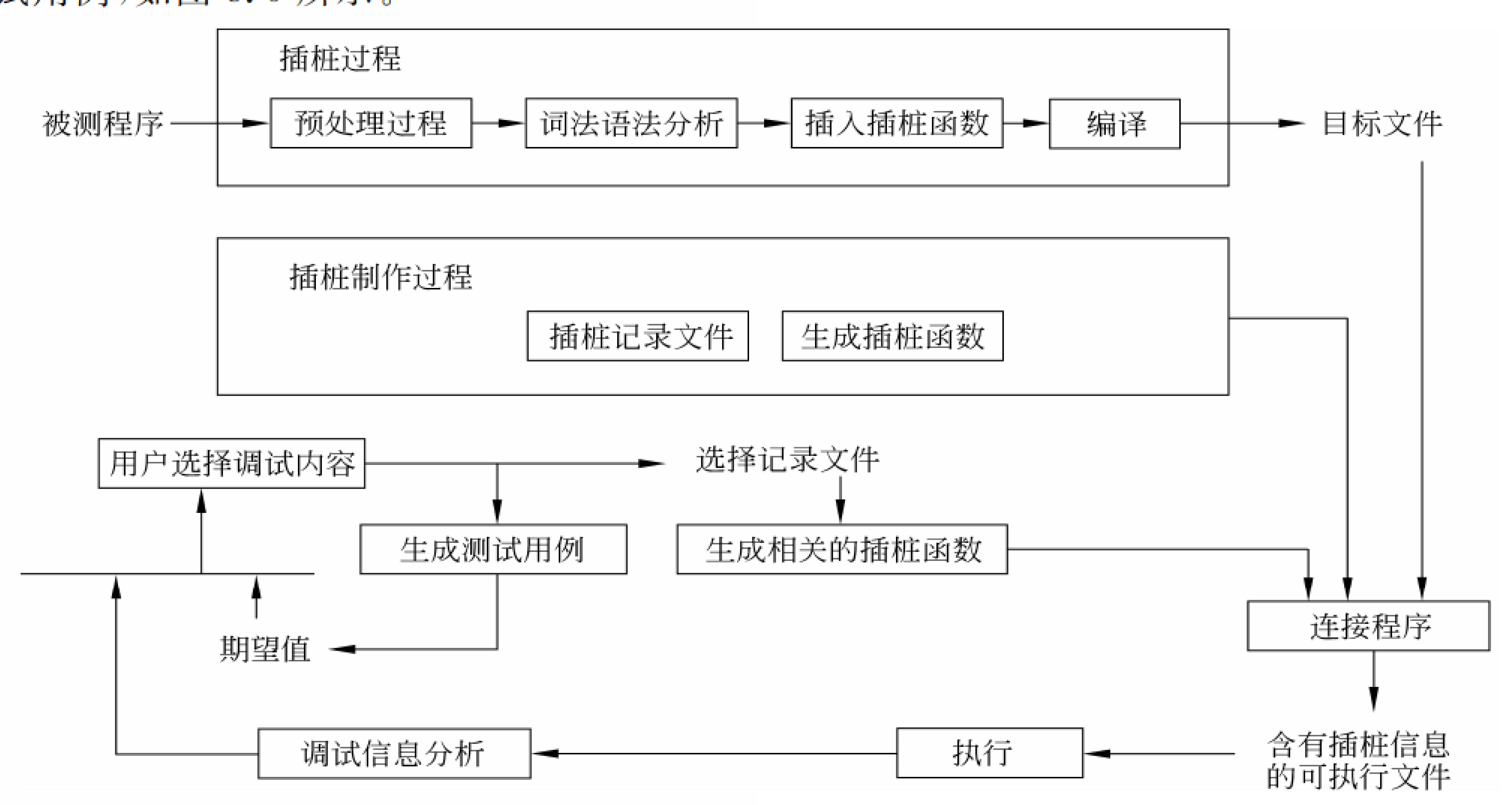


图 8.5 嵌入式软件测试的单元测试

单元测试基本在主机环境上进行，除非特别指定直接在目标环境进行。例如，车载嵌入式系统有的类模块功能与目标环境耦合紧密，单元测试就在目标机上进行。在主机平台上运行测试的速度比在目标平台上快得多，当在主机平台完成测试时，可以在目标环境上重复作一次确认测试，确认测试结果在主机和目标机上没有不同。

2. 集成测试

集成测试在主机平台上模拟目标环境运行，在目标环境上进行重复测试，在此级别上的确认测试将确定一些环境上的问题，如内存定位和分配上的一些错误。在主机环境上的集成测试的使用，依赖于目标系统的具体功能有多少。

3. 确认测试

确认测试把软件系统作为一个执行实体进行的需求有效性测试，其目的是验证软件是否满足所有的功能、行为和执行要求。

4. 硬件软件集成测试

将系统的测试软件系统和硬件、人机交互信息资源和数据库等其他资源结合起来进行测试，确保整个系统的性能和功能都达到了要求，需要在这个阶段和硬件结合，在目标环境中进行测试。

嵌入式软件的交叉测试详细步骤如下：

① 使用测试工具执行静态测试分析，并且为动态覆盖测试准备插装代码。

② 使用源码在主机环境执行功能测试，修正软件的错误和测试脚本中的错误。

③ 插装后的软件执行覆盖率测试，并进行添加测试用例修正软件的错误，保证达到所要求的覆盖率目标。

④ 在目标环境下重复步骤②，确认软件在目标环境中执行的正确性。

⑤ 若测试需要完备，应在目标系统上重复步骤③，确定软件的覆盖率。

8.3 嵌入式软件测试工具

嵌入式软件的测试不但可以使用通用软件的测试技术和测试工具，也有其专门的测试工具，如 CodeTest 等。目前嵌入式软件的测试工具具有纯软件测试工具、纯硬件测试工具和软硬结合的测试工具等。下面依次对这三种测试工具进行介绍。

8.3.1 纯软件测试工具

目前大多数嵌入式测试工具都属于纯软件测试工具，如 Telelogic 公司的 Logiscope、WindRiver 公司的 CoverageScope 等。纯软件测试工具采用软件仿真技术在主机上模拟目标机，使大部分的测试都在主机的仿真机器上运行，采用插桩技术，在被测代码中插入函数或语句，在目标系统中运行预处理任务，将处理后的数据通过目标机的调试口送到宿

主机环境中进行测试。

由于插桩函数和预处理任务的存在，使系统的代码增大，并对系统的运行效率有很大的影响。

8.3.2 纯硬件测试工具

利用万用表、示波器、逻辑分析仪等纯硬的手段进行系统的硬件设计与测试。其中，最常用的纯硬件测试工具是逻辑分析仪。通过监控系统在运行时总线上的指令周期，并以一定的频率捕获这些信号，通过对这些数据分析，了解用户系统的工作状态，判断程序当前运行的状况。由于逻辑分析仪使用的是采样的模式，难免会遗失一些重要的信号，同时，分析的范围也极其有限，往往很难得出满意的结果。

8.3.3 软硬结合测试工具

软硬结合测试工具结合纯软件测试工具和纯硬件测试工具相互各自的优点，如Applied Microsystems Corporation(AMC)公司的CodeTes测试工具。CodeTest不同于纯软件测试工具，没有采用插桩函数而是插入赋值语句，执行时间非常短。CodeTest从纯硬件测试工具里吸取了从总线捕获数据的技术并且进行了改进，不再是采样的方式，而是通过监视系统总线，当程序运行到插入的特殊点的时候才会主动的到数据总线上把数据捕获回来从而执行效率较高。

8.4 习　　题

1. 填空题

(1) 嵌入式系统同通用计算机系统相比具有以下特点：________，________，与外部环境交互的反应性，专用性，体积小，________，并发处理。

(2) 嵌入式软件测试工具划分为________，________，软硬结合的测试工具三类。

(3) 常用的纯软测试工具有________，________等。

(4) 嵌入式软件的测试策略有________，________，________。

(5) 嵌入式软件的测试的四个阶段是________，________，________，________。

2. 论述题

(1) 什么是嵌入式软件？

(2) 简述式软件特点对嵌入式软件测试的影响。

(3) 嵌入式软件测试的关键技术有哪些？

(4) 嵌入式软件有哪些测试策略？

(5) 嵌入式软件测试工具有几类？分别是什么？



软件质量保证

测试项目管理就是以测试项目为管理对象，建立起软件测试管理体系，通过专门的测试组织，运用专门的软件测试知识、技能、工具和方法，对测试项目进行计划、组织、执行和控制，确保软件测试在保证软件质量中发挥关键作用。

下面对软件测试管理、软件测试文档、测试人员组织、软件缺陷管理等方面进行介绍。

9.1 软件测试管理

软件测试管理认为软件测试是一个复杂的系统工程，需要对组成这个系统的各个部分进行识别和管理，实现特定的系统目标。测试系统主要由测试计划、测试设计、测试实施、配置管理、资源管理、测试管理 6 个过程组成。

其中，测试计划、测试设计、测试实施在《软件测试流程》中讲解。测试配置管理作为软件配置管理的子集，作用于测试的各个阶段，其管理对象包括测试计划、测试用例、被测版本、测试工具以及测试环境和测试结果等。资源管理包括人力资源和测试所需的相关技术等管理。测试管理是指采用合适的方法对测试的流程和结果进行监视。

软件测试管理体系一般包括如下 6 个步骤：

(1) 识别软件测试所需的过程及其应用，即测试计划、测试设计、测试实施、配置管理、资源管理、测试管理。

(2) 确定这些过程的顺序和相互作用，前一个的输出作为后一个的输入。其中，配置管理和资源管理作为支撑性的过程。

(3) 确定这些过程所需要的准则和方法，制订 6 个过程所需的文档。

(4) 确保所需的资源和信息，并对 6 个过程进行监测。

(5) 监视、测量和分析这些过程。

(6) 实施必要的过程改进措施。

9.2 软件测试文档

测试文档是对要执行的软件测试和测试的结果进行描述、定义、规定和报告的任何书面或图示信息。由于软件测试是一个复杂的过程，必须把对软件测试的要求、规划、执行过程等有关信息，以及对测试结果的分析、评价，以正式的文档形式给出。

9.2.1 测试文档的类型

根据所起作用的不同，测试文档通常可以分为两类，即前置作业文档和后置作业文档。

1. 前置作业文档

前置作业文档可以使接下来将要进行的软件测试流程更加流畅和规范。测试计划及测试用例的文档属于前置作业文档。测试计划详细规定了测试的要求，包括测试的目的和内容、方法和步骤以及评价测试的准则等。由于要测试的内容可能涉及软件的需求和软件的测试，因此必须及早开始测试计划的编写，至少从需求分析阶段开始。测试用例就是将软件测试的行为和活动做一个科学化组织和归纳，测试用例的好坏决定着测试工作的成功和效率，因此测试用例的选取是做好测试工作的第一步。在软件测试过程中，软件测试行为必须能够加以量化，这样才能进一步让管理层掌握所需要的测试进程，测试用例就是将测试行为和活动具体量化的方法之一。而测试用例文档是为了将软件测试行为和活动转化为可管理的模式，在测试文档编制过程中，按照规定的要求精心设计测试用例有着重要意义。

2. 后置作业文档

后置作业文档是在测试完成后提交的，主要包括软件缺陷报告和分析总结报告。在软件测试过程中，对于发现的大多数软件缺陷，要求测试人员简捷、清晰地把发现的问题以文档形式报告给管理层和判断是否进行修复的小组，使其得到所需要的全部信息，然后决定对软件缺陷是否进行修复及下一步工作。测试分析报告应说明对测试结果的分析情况，经过测试证实了软件具有的功能以及它的缺陷和限制，并给出评价的结论性意见。这个意见既是对软件质量的评价，也是决定该软件能否交付用户使用的一个依据。

IEEE 给出软件测试文档分为测试计划、测试设计规格说明、测试规程规格说明、测试日志、测试缺陷报告和测试总结报告等。下面依次进行介绍。

1）软件测试计划文档

软件测试计划文档主要对软件测试项目以及所需要进行的测试工作、测试人员所应该负责的测试工作、测试过程、测试所需的时间和资源、测试风险、测试项通过/失败的标准、测试中断和恢复的规定、测试完成所提交的材料等做出预先的计划和安排。

2）软件测试设计规格说明文档

软件测试设计规格说明文档用于每个测试等级，以制定测试集的体系结构、通过/失败准则和覆盖跟踪。

3）软件测试用例规格说明文档

软件测试用例规格说明文档用于描述测试用例，包括测试项、输入规格说明、输出规格说明、预期要求和规程需求等。

4）测试规程

测试规程用于指定执行一个测试用例集的步骤。

5）测试日志

测试日志用于记录测试的执行情况不同，可根据需要选用。

6）软件缺陷报告

软件缺陷报告用来描述出现在测试过程或软件中的异常情况，这些异常情况可能存在于需求、设计、代码、文档或测试用例中。

7）测试总结报告

测试总结报告用于报告某个测试的完成情况，给出评价和建议。

9.2.2　测试文档的重要性

测试文档的重要性主要表现在如下几个方面：

1. 验证需求的正确性

测试文件中规定了用以验证软件需求的测试条件，由于要测试的内容可能涉及软件的需求和软件的设计，因此必须及早开始测试计划的编写工作。通常，测试计划的编写从需求分析阶段开始，到软件设计阶段结束时完成。

2. 检验测试资源

测试计划不仅要用文件的形式把测试过程规定下来，还应说明测试工作必不可少的资源，进而检验这些资源是否可以得到，即它的可用性如何。

3. 明确任务的风险

测试计划文档帮助测试人员分析测试可以做什么，不能做什么。了解测试任务的风险有助于对潜伏的可能出现的问题事先作好思想上和物质上的准备。

4. 生成测试用例

测试用例的好坏决定着测试工作的效率，选择合适的测试用例是作好测试工作的关键。在测试文件编制过程中，按规定的要求精心设计测试用例有重要的意义。

5. 评价测试结果

测试文件包括测试用例，即若干测试数据及对应的预期测试结果。完成测试后，将测试结果与预期的结果进行比较，便可对已进行的测试提出评价意见。

6. 确定测试的有效性

完成测试后，把测试结果写入文件，这对分析测试的有效性，甚至整个软件的可用性提供了依据。同时还可以证实有关方面的结论。

9.3 测试人员组织

9.3.1 测试团队架构

软件测试团队组织管理通俗地讲就是测试团队应该如何组建。由于测试过程与测试团队的架构管理密不可分。测试过程组织的框架如图 9.1 所示。

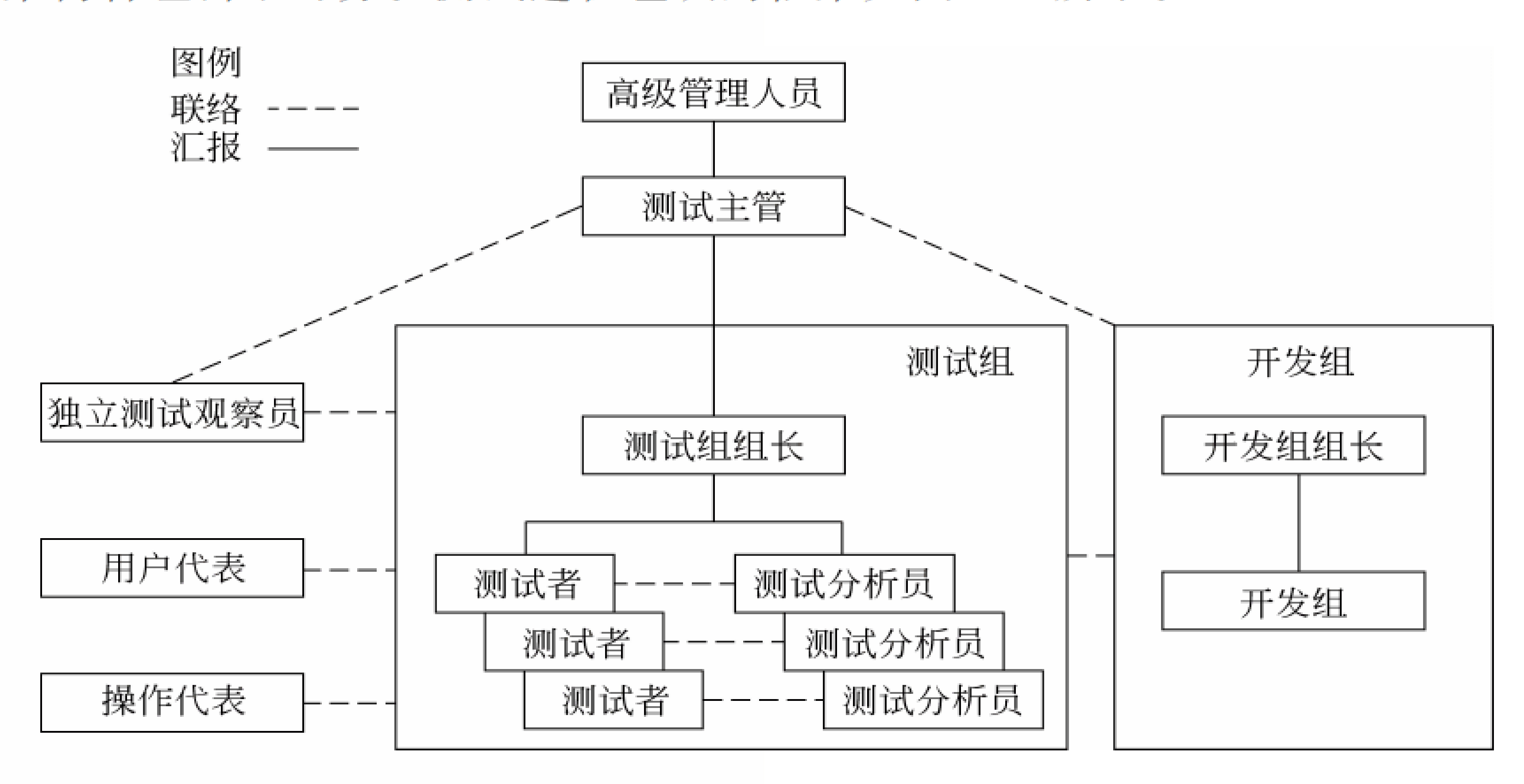


图 9.1 测试过程组织

1. 测试主管

测试主管有权管理测试过程日常的组织，负责保证在给定的时间、资源和费用的限制下行个测试项目产生满足所需的质量标准的产品。测试主管负责与开发组联系，保证单元测试的顺利进行，并与独立测试观察员联系，接收有关没有正确遵循测试过程的测试项目的报告。

测试主管向公司内的高级主管或领导报告，如质量保证主管或信息技术领导。在大的公司中，尤其对于那些遵循规范的项目管理过程的公司中。测试主管可以向测试程序委员会报告，该委员会负责把握测试程序的项目管理的总体方向。

2. 测试组组长

测试组组长负责为测试分析员和测试者分配任务，按照预定的计划监控他们的工作进度，建立和维护测试项目文件系统，保证产生测试项目相关材料（测试计划文档、测试规范说明文档），测试组组长负责产生这个文档也可以授权测试分析员来完成这个文档。

测试组组长听取一个或多个测试分析员的测试报告，在验收测试时，测试组组长负责和用户代表、操作代表联系. 以便有一个或多个用户来执行用户和操作验收测试。

3. 测试分析员

测试分析员负责设计和实现用于完成自动化测试的一个或多个测试脚本，协助测试组组长生成测试规格说明文档。

在调试测试用例的设计过程中，测试分析员需要分析自动化测试的需求规格说明，以便确定必须测试的特定需求。在这个过程中，测试分析员应该优先考虑测试用例，以反映被确认的特性的重要性以及在正常使用自动化测试中导致失败的特性的风险。在完成测试项目后，测试分析员负责备份和归档所有的测试文档和材料。这些材料将提交给测试组组长进行归档。测试分析员还负责完成一份测试总结报告。

4. 测试者

测试者主要负责执行由测试分析员建方的测试脚本，并负责解释测试用例结果并将结果记录到文档中。

在执行测试脚本之前，测试者首先要建立和初始化测试环境，其中包括测试数据和测试硬件，以及其他支持测试所需的软件。在测试执行过程中，测试者负责填写测试结果记录表格，以便记录执行每个测试脚本观察到的结果。测试者使用测试脚本对预期结果进行描述。在完成测试以后，测试者还负责备份测试数据、模拟器或测试辅助程序以及测试中使用的硬件的说明。这些材料将提交给测试组组长归档。

9.3.2 测试团队阶段性

为了保证软件的开发质量，软件测试应贯穿于软件定义与开发的整个过程。因此，对于软件开发中的分析、设计和实现等各个阶段所得到的结果，都应进行软件测试。在不同的阶段测试团队也不尽相同，体现了测试团队的阶段性。

1. 需求分析阶段

需求分析规格说明是否完整、正确、清晰是软件开发成败的关键，因此，为了确保需求的质量，应对其进行严格的审查。测试评审小组通常可有一名组长和若干成员组成，其成员包括系统分析员，软件开发管理者，软件设计、开发、测试人员和用户。

2. 设计阶段

软件设计是将软件需求转换成软件表示的过程。主要描绘出系统结构、详细的处理过程和数据库模式。按照需求的规格说明对系统结构的合理性、处理过程的正确性进行评价，利用关系数据库的规范化理论对数据库模式进行审查。测试评审小组的组成：组长一名，成员包括系统分析员、软件设计人员、测试负责人员。

3. 测试阶段

软件测试是软件质量保证的关键。通常，在编写出每个模块之后，进行单元测试，之后需要对软件系统进行各种综合的测试。测试评审小组包括组长一名，负责整个测试的

计划、组织工作；以及具备一定分析、设计与编程经验的测试组成员，人数可随具体情况确定，一般为3～5人。

9.4 软件缺陷管理

9.4.1 概述

1.2节讲解了软件缺陷案例、特征等。

软件缺陷具有如下内容：

（1）对缺陷的描述应该包含可追踪信息。

如给每个缺陷分配一个缺陷号。每个编号必须是唯一的，可以根据该编号搜索、根据、查看该缺陷的处理情况。

（2）对缺陷的描述应该包含缺陷的基本信息。

通常缺陷的基本信息包括缺陷状态、缺陷标题、缺陷严重程度、缺陷紧急程度、缺陷提交人、缺陷提交日期、缺陷所属、缺陷解决人、缺陷解决时间、缺陷解决结果、缺陷处理人、缺陷处理最终时间、缺陷处理结果、缺陷确认人、缺陷确认时间、缺陷确认结果等。

具体如下所示：

① 缺陷状态：标注缺陷待修正、待评审、待验证、关闭等状态信息。

② 缺陷标题：简明地说明缺陷的类型及内容。

③ 缺陷严重程度：测试人员给出的缺陷严重程度估计，可以是致命的、严重的、一般的、建议的。

④ 缺陷紧急程度：测试人员给出的测试处理优先级。

⑤ 缺陷提交人：发现此缺陷的测试人员，最好附有联系方式，以方便缺陷处理人员进行确认。

⑥ 缺陷提交日期：提交人提交缺陷的日期。

⑦ 缺陷所属：指缺陷所在的模块或者是缺陷所属的开发文档的名称。

⑧ 缺陷解决人：由谁来进行缺陷的解决，明确是需求分析人员、设计人员还是程序编码人员。

⑨ 缺陷解决时间：项目组负责人返回的缺陷预计处理的时间。

⑩ 缺陷解决结果：预计缺陷修改后能达到的结果。

⑪ 缺陷处理人：应该由谁来处理这个缺陷。

⑫ 缺陷处理最终时间：指缺陷得到处理的实际时间。

⑬ 缺陷处理结果：缺陷最后的实际处理结果。

⑭ 缺陷确认人：由谁来确认缺陷已经得到了修正。

⑮ 缺陷确认时间：缺陷修复的确认工作完成的时间。

⑯ 缺陷确认结果：确认软件缺陷的修正工作是否有效。

（3）对缺陷的描述应该包含缺陷的详细描述。

应对缺陷的特征应做详细的描述，例如程序代码中的错误，应详细描述错误发生的

软硬件环境，相关输入输出数据，出错时程序的状态等等，以方便编码人员进行错误复现和错误定位。又如设计规格说明中的错误，应指明它与高层软件开发文档(如需求规格说明)中哪些条款相违背，为什么判为违背，都需要描述清楚，以方便设计人员进一步核实。

9.4.2 缺陷跟踪流程

在软件的测试或评审过程中，为了不遗漏任何缺陷，并提高缺陷修复工作的质量，通常需要执行缺陷跟踪。所谓缺陷跟踪是指从缺陷被发现开始到被改正为止的整个跟踪流程，如图 9.2 所示。

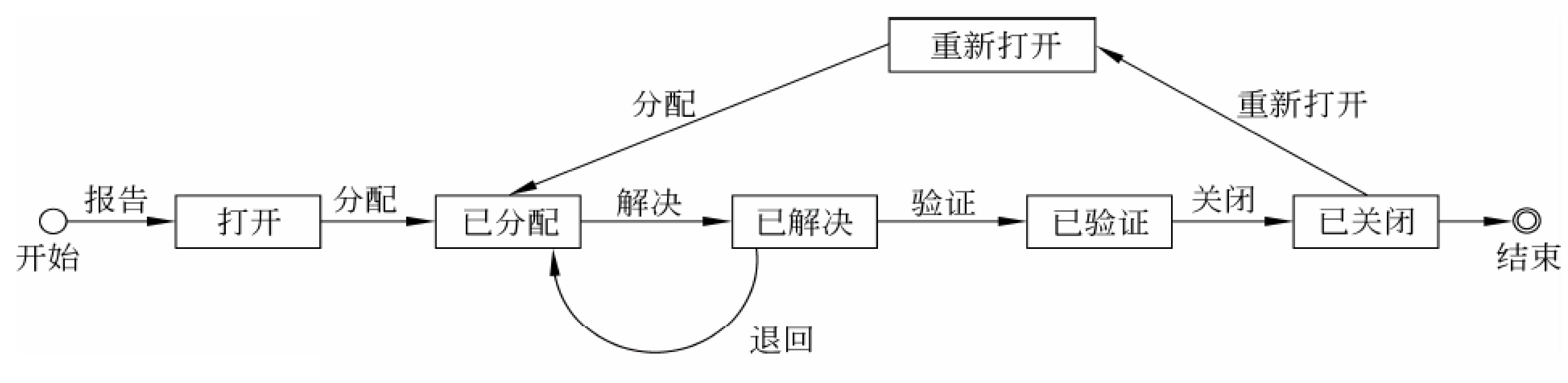


图 9.2 典型的缺陷跟踪流程

图 9.2 是一个状态转换图，每个圆角矩形表示缺陷的一个状态，箭头线表示引起缺陷状态变化的事件。一个缺陷被报告后，其状态被设置为“打开”，它被分配给一个开发人员进行修复，此时缺陷状态设置为“已分配”，然后开发人员开始修复缺陷，修复完毕后，将缺陷状态设置为“已解决”，此时测试人员可以开始回归测试，如果回归测试通过，确认缺陷已被修复，则将缺陷处理状态设置为“已验证”，否则退回给开发人员重新进行修复。一个缺陷结束了其生命周期后，可将其关闭，其处理状态变为“已关闭”。已被关闭的缺陷如果在某些情况下被发现仍有问题，可以将其重新打开，使其处理状态变为“重新打开”，以便再分配给开发人员进行修复。

该流程涉及测试人员、项目负责人、开发人员和评审员的角色，具体如下所示。

- 测试人员：执行测试的人，是缺陷的报告者，负责报告缺陷或确认缺陷是否可以“提交”、“未通过”、“通过”。
- 项目负责人：对整个项目负责，对产品质量负责，需要及时了解当前有哪些新的缺陷，哪些必须及时修正，“分配”任务。
- 开发人员：设计和编码人员，用于了解哪些缺陷需要“修正”与“不修正”。
- 评审员：对缺陷进行最终确认，行使仲裁权力。

9.4.3 缺陷跟踪管理系统概述

缺陷跟踪管理系统是用于集中管理软件测试过程中发现的缺陷的数据库程序，可以通过添加、修改、排序、查寻、存储操作来管理软件缺陷。缺陷跟踪管理系统具有如下的作用。

1. 便于缺陷的查找和跟踪

对于大中型软件的测试过程而言，报告的缺陷总数可能会达成千上万个，如果没有缺陷跟踪管理系统的支持，要求查找某个错误，其难度和效率可想而知。

2. 便于协同工作

缺陷跟踪管理系统可以作为测试人员、开发人员、项目负责人、缺陷评审人员协同工作的平台。

3. 保证测试工作的有效性

避免测试人员重复报错，同时也便于及时掌握各缺陷的当前状态，进而完成对应状态的测试工作。

4. 便于跟踪和监控错误的处理过程和方法

方便地检查处理方法是否正确，跟踪处理者的姓名和处理时间，作为工作量的统计和业绩考核的参考。

9.5 软件质量

9.5.1 概述

在《辞海》和《辞源》中，质量解释为"产品或工作的优劣程度"。软件质量具有多种定义。ANSI/IEEE Std 729-1983 定义软件质量为"与软件产品满足规定的和隐含的需求的能力有关的特征或特性的全体"。CMM 对质量的定义是：

（1）一个系统、组件或过程符合特定需求的程度。

（2）一个系统、组件或过程符合客户或用户的要求或期望的程度。

M. J. Fisher 定义软件质量为"所有描述计算机软件优秀程度的特性的组合"。

人们把影响软件质量的特性用软件质量模型来描述，从正确性、可靠性、健壮性、美观性、性能、易用性、兼容性、安全性、可移植性、可扩展性等特性保证软件的质量。具体如下所示：

（1）正确性是指软件按照需求正确执行任务的能力，涵盖了精确性。

（2）可靠性是指在一定的环境下，在给定的时间内，系统能够正常运行的概率。

（3）健壮性是指在异常或者不利情况下，软件能够正常运行的能力。

（4）美观性主要指软件用户界面设计的情况，美观性就是从大众化审美以及心理学角度对软件提出的要求，包括软件的颜色搭配，字体使用，排版布局等方面。

（5）性能也就是一个软件效率问题，也就是软件特定时间空间环境下系统的响应能力。

（6）易用性是软件能否满足客户容易操作使用程度，是衡量一款软件质量好坏的一

个重要方面。

(7) 兼容性指一款软件和其他不同软件通信(或交换信息)的能力。

(8) 安全性是指软件系统防止被非法入侵的能力。

(9) 可移植性指的是软件不经修改或稍加修改就可运行于不同软硬件环境(CPU、OS 和编译器)的能力,主要体现为代码的可移植性。

(10) 可扩展性反映软件适应"变化"的能力,如增加新功能等。可扩展性和可移植性一样,主要都是从开发的角度对软件提出的要求。

当前,软件过程的质量管理评估标准主要有三大体系:ISO9000、CMM/CMMI 和 ISO15504 等。下面逐一详细介绍。

9.5.2 ISO 9000 系列

自从 1987 年 ISO 9000 族标准公布以来,其已经成为全球最有影响的质量管理和质量保证标准。ISO 9000 族标准的制订和实施反映了市场经济条件下供需双方在进行交易活动中的要求。供方只要按 ISO 9000 族标准组织产品的开发和生产,并通过权威机构的认证,在产品质量方面就会赢得顾客的充分信任。需方在市场上选购产品时,更愿意选择通过质量认证的企业所生产的产品,从而减少质量的检验活动。

ISO 9000 系列标准原本是为制造硬件产品而制定的标准,不能直接用于软件制作。为了应用于软件企业,制定出 ISO 9000-3 标准,全称为"在计算机软件开发、供应、安装和维护中的使用指南",其核心思想是软件产品的质量取决于软件生存期所有阶段的活动。ISO 9000-3 的要点包括以下几个方面:

(1) ISO 9000-3 标准仅适合于依照合同进行的单独的订货开发软件,不适用于面向多数用户销售的程序软件包。

(2) 对于包括合同在内的全部工序要进行审查,并要求一切文档化。

(3) ISO 9000-3 对合同双方的责任均做出了明确规定,需方应收集供方意见,归纳形成需方需求,详细传达给供方,才可能对供方提成实施质量保证的要求。

(4) 软件在完成设计编码后,测试和验收对提高软件质量是很有限的,必须建立质量保证体系,全面管理和控制软件生存期所有阶段的质量活动。

ISO/IEC 9126 是软件产品评估—质量特性及其使用指南纲要,是作为软件产品质量的大范围律定及评估,确保质量充分的重要因素。ISO/IEC 9126 标准中,定义了六种质量特性,并且描述了软件产品评估过程的模型。ISO/IEC 9126 第一部分所定义的软件质量特性,可用来指定客户及使用者在功能性与非功能性方面的需要。

ISO9126(GB/T 16260)《信息技术软件产品质量》,描述新的软件质量模型,修订成 4 个部分,如图 9.3 所示。

(1) ISO 9126-1:2001 第 1 部分:质量模型;

(2) ISO 9126-2:2003 第 2 部分:外部质量度量;

(3) ISO 9126-3:2003 第 3 部分:内部质量度量;

(4) ISO 9126-4:2004 第 4 部分:使用质量度量。

ISO/IEC9126 软件质量模型是一种评价软件质量的通用模型,包括质量特性、质量

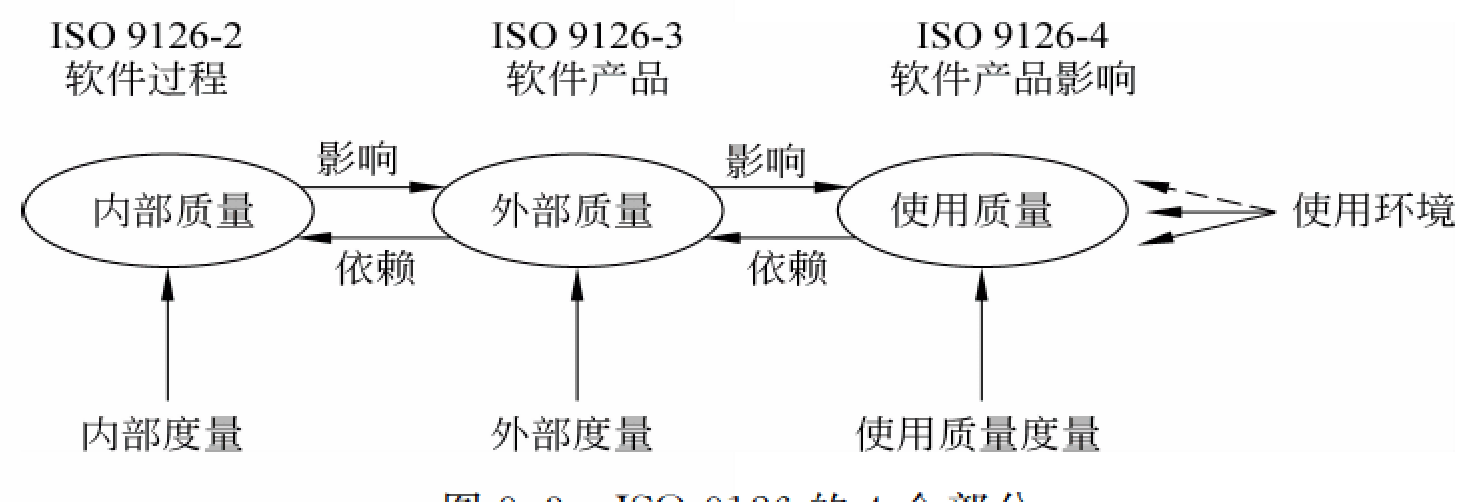


图 9.3 ISO 9126 的 4 个部分

子特性和度量指标 3 个特性。其中,质量特性包括:功能性、可靠性、易使用性、效率、可维护性和可移植性。在质量子特性中,安全性子特性属于功能性;成熟性、容错性、易恢复性属于可靠性;易分析性、易改变性、易测试性、稳定性属于可维护性;适用性属于可移植性。

9.5.3 CMM/CMMI

1987 年 9 月,卡内基-梅隆大学的软件工程研究所谓美国国防部开发了软件过程评估方法和能力成熟度模型 CMM。该模型有效地帮助软件公司建立和实施过程改进计划,用来定义和评价软件公司开发过程的成熟度,为提高软件质量提供指导。

CMM 为软件企业的过程能力提供了一个阶梯式的进化框架,该框架共有 5 级,分别是初始级、可重复、已定义级、已管理级和优化级。第一级实际上是一个起点,任何准备按 CMM 体系进化的企业都自然处于这个起点上,并通过这个起点向第二级迈进。

1. 初始级

在这个阶段,软件开发过程表现得非常随意,偶尔会出现混乱的现象,只有很少的工作过程是经过严格定义的,开发成功往往依靠的是某个人的智慧和努力。此时的软件机构基本没有健全的软件工程管理制度,其软件过程完全取决于项目组的人员配备,具有不可预测性。人员变了过程也随之改变,软件过程是不稳定的,产品质量只能根据相关人员的个人工作能力而不是软件机构的过程能力来预测。

2. 可重复级

此阶段已经建立了基本的项目管理过程。按部就班地设计功能、跟踪费用,根据项目进度表进行开发。对于相似的项目,可以重用以前已经开发成功的部分。处于 2 级成熟度的软件机构,针对所承担的软件项目已建立了基本的软件管理控制制度。通过对以前项目的观察和分析,可以提出针对现行项目的约束条件,软件机构已经制定了项目标准,并且能确保严格执行这些标准。软件项目的策划和跟踪是稳定的,已经为一个有秩序的管理过程提供了可重复以前成功实践的项目环境。软件项目工程活动处于项目管理体系的有效控制之下,执行着基于以前项目的准则且合乎现实的计划。

3. 已定义级

在此阶段,软件开发的工程活动和管理活动都是文档化、标准化的,是被集成为一个

有组织的标准开发过程，所有项目的开发和维护都在这个标准基础上进行定制。处于3级成熟度的软件机构，无论是管理活动还是工程活动都是稳定的。软件开发的成本和进度以及产品的功能和质量都受到控制，而且软件产品的质量具有可追溯性。这种能力是基于在软件机构中对已定义的过程模型的活动、人员和职责都有共同的理解。

4. 已管理级

此阶段的软件过程是可度量的，软件过程在可度量的范围内运行。软件发布时间由事先确定的指标决定，软件在没有达到目标之前不能发布。软件的开发在发生偏离时可以及时采取措施予以纠正，并且可以预期软件产品是高质量的。

5. 优化级

此阶段通过建立开发过程的定量反馈机制，不断产生新的思想，采用新的技术来优化开发过程。处于5级成熟度的软件机构，可以通过对过程实例性能的分析和确定产生某一缺陷的原因，来防止再次出现这种类型的缺陷，通过对任何一个过程实例的分析所获得的经验教训都可以成为该软件机构优化其过程模型的有效依据，软件过程是可优化的。此级的软件机构能够持续不断地改进其过程能力，既对现行的过程实例不断地改进和优化，又借助于所采用的新技术和新方法来实现未来的过程改进。

CMM不同成熟度等级过程的可视性和过程能力如表9.1所示。

表9.1 CMM不同成熟度等级过程的可视性和过程能力

能力等级	特　点	可视性	过程能力
初始级	软件过程是混乱无序的，对过程几乎没有定义，成功依靠的是个人的才能和经验，管理方式属于反应式	有限的可视性	一般达不到进度和成本的目标
可重复级	建立了基本的项目管理来跟踪进度、费用和功能特征，制定了必要的项目管理，能够利用以前类似的项目应用取得成功	里程碑上具有管理可视性	由于基于过去的性能，项目开发计划比较现实可行
已定义级	已经将软件管理和过程文档化、标准化，同时综合成该组织的标准软件过程，所有的软件开发都使用该标准软件过程	项目定义软件过程的活动具有可视性	基于已定义的软件过程，组织持续地改善过程能力
已管理级	收集软件过程和产品质量的详细度量，对软件过程和产品质量有定量的理解和控制	定量地控制软件过程	基于对过程和产品的度量，组织持续地改善过程能力
优化级	软件过程的量化反馈和新的思想和技术促进过程的不断改进	不断地改善软件过程	组织持续地改善过程能力

总而言之，根据软件生产的历史与现状，CMM框架可用5个不断进化的等级来表达：其中初始级是混沌的过程；可重复级是经过训练的软件过程；已定义级是标准一致的软件过程；可管理级是可预测的软件过程；优化级是能持续改善的软件过程。

CMMI包括软件工程、系统工程和软件采购等在内的模型集成，以解决除软件开发以外的软件系统工程和软件采购工作中的需求。CMMI模型如图9.4所示。

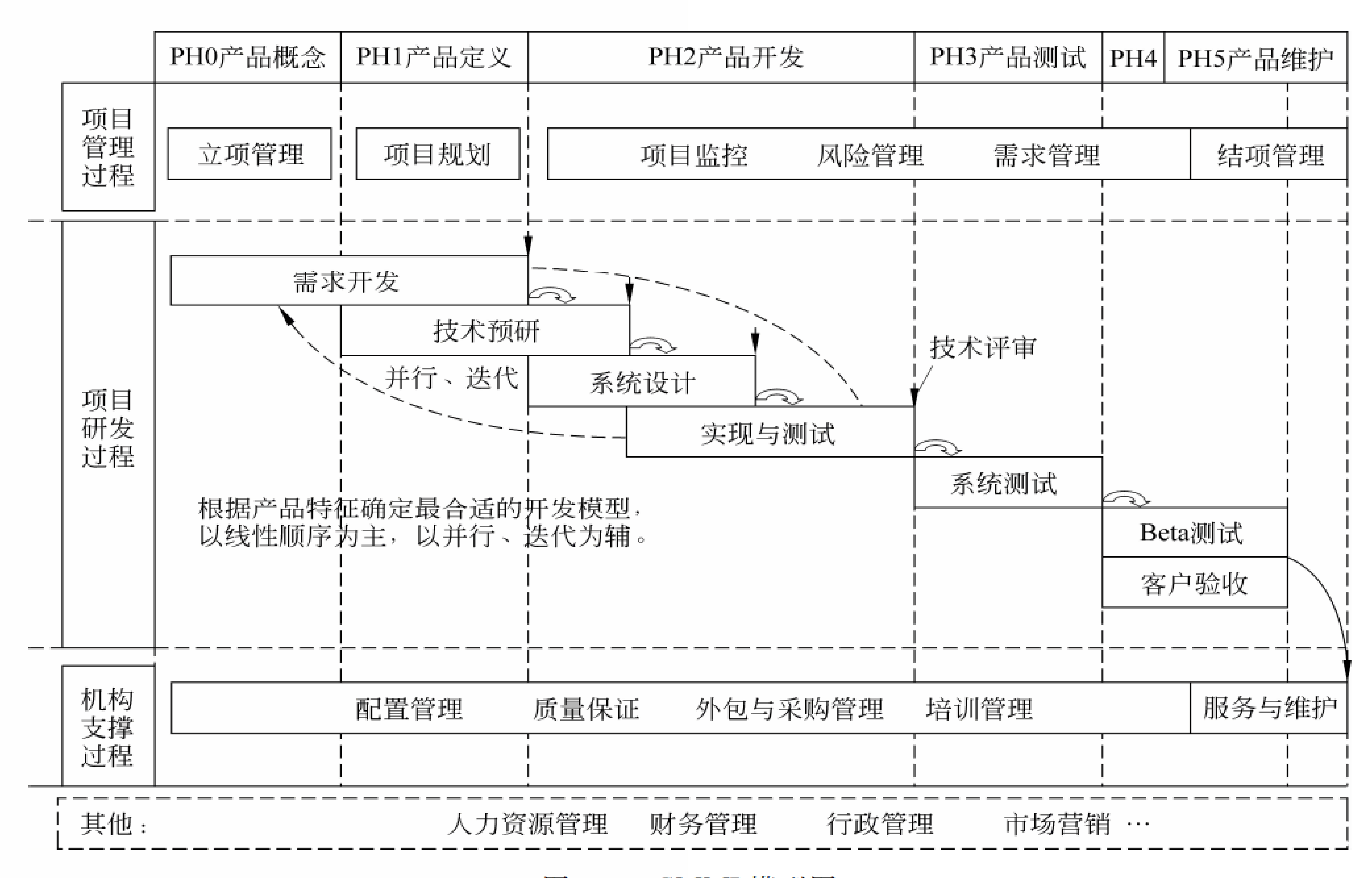


图 9.4　CMMI 模型图

CMMI 纠正了 CMM 的一些缺点，消除了不同模型之间的不一致和重复，降低基于模型改善的成本，指导组织改善软件过程，提高产品和服务的开发、获取和维护能力。

CMM 和 CMMI 作为软件质量的保证措施，其目标是为提高组织过程和管理产品开发、发布和维护的能力提高保障，帮助组织客观地评价自身能力成熟度和过程域能力，为过程改进建立优先级以及执行过程改进，归根结底也是为了提高软件产品的质量。CMM/CMMI 的质量管理理念是“产品的质量在很大程度上取决于用以开发和维护该产品的过程的质量”。因此，CMM/CMMI 排行基于过程的软件质量管理，质量管理活动是CMM/CMMI 实施过程中的核心内容，无论是质量保证、度量与分析、验证、确认、质量管理等过程域，无一不贯穿于整个 CMM/CMMI 的实施全过程。

通过实施 CMM/CMMI，有助于改进软件产品的质量、改进项目满足预定目标能力、减少开发成本和周期，降低项目风险、提高组织过程能力，提高市场占有率。实施不同等级的 CMM/CMMI，对于按照每功能点来计算的软件缺陷率的降低具有显著的作用。CMM/CMMI 基于过程的软件质量管理主要包括质量保证和质量控制两大方面。软件质量保证是由（相对）独立的质量管理人员在项目的整个开发周期中对项目所执行的过程和产生的工作产品进行监督和检查，确保其符合预定的要求。软件质量保证的目的是确保过程得到有效的执行，并推进过程改进，并就项目过程的执行情况和所构造的产品向管理者提供适当的可视性。软件质量控制是指为评价和验证已开发的产品而执行的活动和技术，包括验证产品是否满足质量要素的要求，以及产品（包括生命周期的工作产品）是否具有接受的质量。

软件质量控制所采取的主要技术是软件测试，通过软件测试，来验证产品是否符合技术文档预期的特性、功能和性能等要求，并识别产品的缺陷。

9.5.4 ISO 15504 过程评估

20 世纪 90 年代初，ISO/IEC 是国际标准化组织（ISO）和国际电工委员会（IEC）联合组建的第一个标准化技术委员会，注意到软件过程改进和评估的重要性以及由于缺乏统一的国际标准给软件产业造成的困境，于 1993 年发起了制订 ISO/IEC 15504 系列标准的前期工作。项目名称是“软件过程改进和能力测定”（Software Process Improvement and Capability Determination，SPICE）。

SPICE 项目有三个主要目的：

（1）为软件过程评估标准拟订草稿。

（2）根据草稿进行试验。

（3）努力推动软件产业界过程评估。

1994 年，SPICE 项目的基准文件出台。试验分三个阶段进行，第一阶段：从 1994 年至 1996 年 9 月，主要目的是对文件的关键部分进行验证，包括过程管理模型、实施评估指南、评分过程需求、评估工具构建、选择指南。全球共有 35 个项目参加了第一阶段的试验。第二阶段：从 1996 年 9 月至 1998 年 10 月，全球各地共有几百个项目参加了试验，目的是：评价全部基准文件的实用性和一致性；评价过程管理模型能否体现软件工程和管理的基础实践；评价评估结果的可重复性；评价文件要求的正确性；评价过程能力测定指南的可使用性；评价过程改进指南的可使用性；评价在不同环境中评估框架的可移植性。第三阶段：从 1998 年 10 月至今，目的是验证 SPICE 的总体目标和标准的需求，由于 ISO/IEC 15504 TR（Technology report，技术报告）已经发布，本阶段 SPICE 试验的一个重要目的是为修改 ISO/IEC 15504 TR，将其上升为正式的国际标准提供依据。

9.6 习　　题

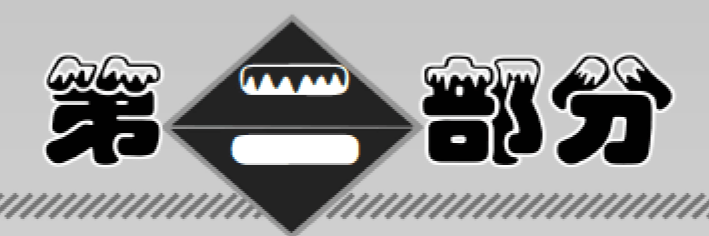
（1）简述软件测试管理内容。

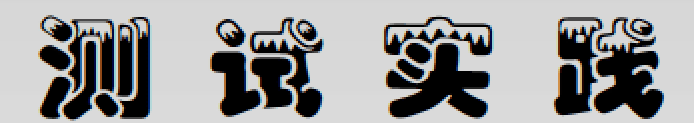
（2）测试文档有哪些作用？主要的软件测试文档有哪些？

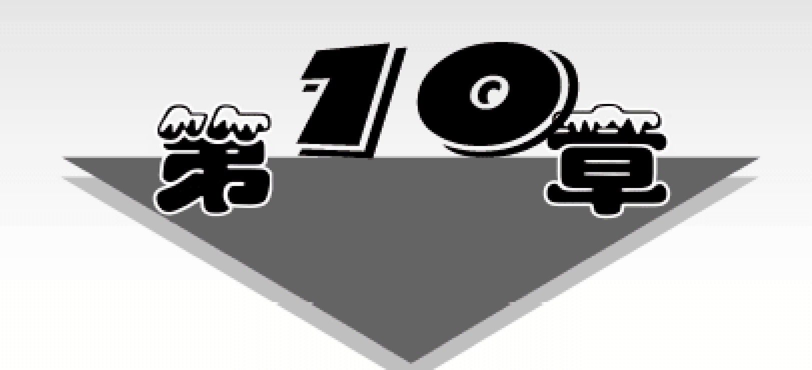
（3）如何理解测试人员的不同的角色划分？

（4）缺陷管理的主要内容是什么？

（5）软件过程的质量管理评估标准主要有三大体系，各是什么？


第二部分


测试实践



软件测试自动化与测试工具

10.1 自动化测试

随着软件变得越来越庞大和越来越复杂，软件测试的工作量也随之增大。由于手工测试的局限性，以下场合往往会优先考虑使用自动化测试。

（1）软件需求变动不频繁。

当软件需求变动过于频繁，势必多次更新测试用例以及测试脚本，增大维护脚本的成本。一般情况，对于需求中相对稳定的模块进行自动化测试，变动较大的模块采用手工测试。

（2）项目周期足够长。

由于自动化测试需求的确定、自动化测试框架的设计、测试脚本的编写与调试需要相当长的时间来完成。

（3）自动化测试脚本可重复使用，手工测试完成难度较大的场合。

性能测试、压力测试、负载测试等需要模拟大量并发用户时，自动化测试脚本可重复使用。许多与时序、死锁、资源冲突、多线程等有关的软件缺陷很难通过手工测试的场合。

（4）回归测试。

回归测试是软件每次有新版本都必须执行，也就是在软件的生命周期中会被反复执行的测试，因此这类测试很适合自动化测试。

软件测试工具实现手工测试无法实现的功能，减轻了手工测试的工作量，减少了测试的执行时间，提高了测试效率。当然，软件测试工具也有如下不足：

（1）某些测试工具难于学习和使用，创建和修改测试脚本费时费力。

（2）测试工具根据实际需要确定是否选用和选用什么样的测试工具，只能解决某一方面问题，应用范围狭窄。

自动化测试发展经历了机械方式实现人工重复操作、统计分析的自动测试、面向目标的自动测试技术和智能应用的自动测试技术等四个阶段，如图10.1所示。

第一阶段：实现人工重复操作。

最初，自动化测试主要研究如何采用自动方法来实现和替代人工测试中的繁琐和机械重复的工作。此时的自动测试活动只是软件测试过程的偶然行为，在一定程度上提高效率，简化测试人员工作，但对整体的测试过程并无太大改进。

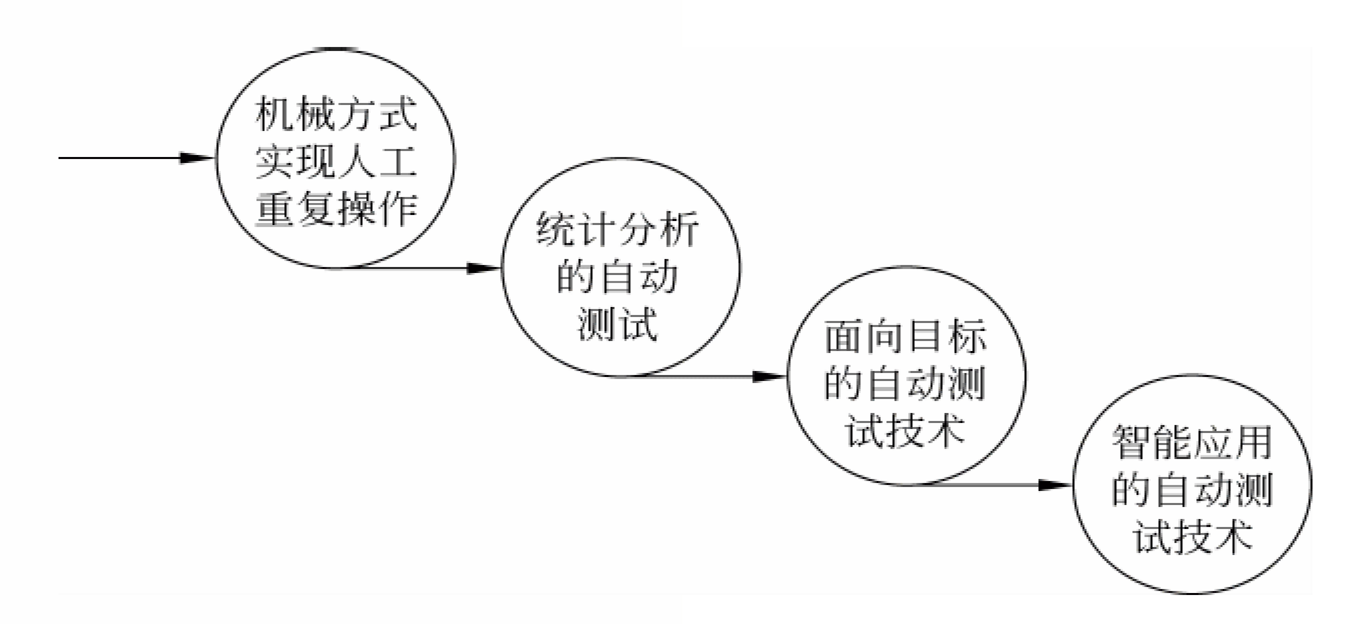


图 10.1 自动化测试发展阶段

第二阶段：统计分析的自动测试。

此阶段的测试针对不同的测试准则和测试策略，指导测试的自动化过程以及对测试的结果进行评估。

第三阶段：面向目标的自动测试技术。

由于进化计算和人工智能等技术以及各种高性能的算法引入到自动测试技术，使得自动测试技术具有目标性。

第四阶段：智能应用的自动测试技术。

测试业界产生测试成熟度模型，针对不同的自动测试具有不同的等级评价标准。

10.2 测试成熟度模型

由于 CMM 没有充分的定义测试，没有提及测试成熟度，没有对测试过程改进进行充分说明，在 KPA 中没有定义测试问题，与质量相关的测试问题，如可测性、充分测试标准、测试计划等方面也没有阐述。测试成熟度模型（Testing Capability Maturity Model，TMM）受 CMM 模型启发产生，TMM 描述了测试的过程，分为初始级、定义级、集成级、管理和测量级和优化，预防缺陷和质量控制级 5 个等级。

1. 初始级

TMM 初始级软件测试过程的特点是测试过程无序，有时甚至是混乱的，几乎没有妥善定义的。初始级中软件的测试与调试常常被混为一谈，软件开发过程中缺乏测试资源，工具以及训练有素的测试人员。初始级的软件测试过程没有定义成熟度目标。

2. 定义级

在 TMM 的定义级中，测试已具备基本的测试技术和方法，软件的测试与调试已经明确地被区分开。这时，测试被定义为软件生命周期中的一个阶段，它紧随在编码阶段之后，由于测试计划往往在编码之后才得以制订，这显然有悖于软件工程的要求。

TMM 的定义级中需实现 3 个成熟度目标：制订测试与调试目标，启动测试计划过程，制度化基本的测试技术和方法。

1）制订测试与调试目标

软件组织必须区分软件开发的测试过程与调试过程，识别各自的目标、任务和活动。正确区分这两个过程是提高软件组织测试能力的基础。与调试工作不同，测试工作是一种有计划的活动，可以进行管理和控制。这种管理和控制活动需要制订相应的策略和政策，以确定和协调这两个过程。

制订测试与调试目标包含5个子成熟度目标：

（1）分别形成测试组织和调试组织，并有经费支持。

（2）规划并记录测试目标。

（3）规划并记录调试目标。

（4）将测试和调试目标形成文档，并分发至项目涉及的所有管理人员和开发人员。

（5）将测试目标反映在测试计划中。

2）启动测试计划过程

测试计划作为过程可重复、可定义和可管理的基础，包括测试目的、风险分析、测试策略以及测试设计规格说明和测试用例。此外，测试计划还应说明如何分配测试资源，如何划分单元测试、集成测试、系统测试和验收测试。启动测试计划过程包含5个子目标：

（1）建立组织内的测试计划组织并予以经费支持。

（2）建立组织内的测试计划政策框架并予以管理上的支持。

（3）开发测试计划模板并分发至项目的管理者和开发者。

（4）建立一种机制，使用户需求成为测试计划的依据之一。

（5）评价、推荐和获得基本的计划工具并从管理上支持工具的使用。

3）制度化基本的测试技术和方法

改进测试过程能力，组织中应用基本的测试技术和方法，并说明何时和怎样使用这些技术、方法和支持工具，基本测试技术和方法制度化有如下两个子目标：

（1）在组织范围内成立测试技术组，研究、评价和推荐基本的测试技术和测试方法，推荐支持这些技术与方法的基本工具。

（2）制订管理方针以保证在全组织范围内一致使用所推荐的技术和方法。

3. 集成级

TMM的集成级中，测试不再是编码阶段之后的阶段，已被扩展成与软件生命周期融为一体的一组活动。测试活动遵循V字模型。测试人员在需求分析阶段便开始着手制定测试计划，根据用户需求建立测试目标和设计测试用例。软件测试组织提供测试技术培训，测试工具支持关键测试活动。但是，集成级没有正式的评审程序，没有建立质量过程和产品属性的测试度量。

集成级要实现如下4个成熟度目标：建立软件测试组织，制订技术培训计划，软件生命周期测试，控制和监视测试过程。

1）建立软件测试组织

软件测试过程对软件产品质量有直接影响。由于测试往往是在时间紧、压力大的情况下完成一系列复杂活动，测试组完成与测试有关的活动，包括制订测试计划，实施测试

执行，记录测试结果，制订与测试有关的标准和测试度量，建立测试数据库，测试重用，测试跟踪以及测试评价等。

建立软件测试组织要实现 4 个子目标：

（1）建立全组织范围内的测试组，并得到上级管理层的领导和各方面的支持，包括经费支持。

（2）定义测试组的作用和职责。

（3）由训练有素的人员组成测试组。

（4）建立与用户或客户的联系，收集他们对测试的需求和建议。

2）制订技术培训计划

为高效率地完成好测试工作，测试人员必须经过适当的培训。

制订技术培训规划有 3 个子目标：

（1）制订组织的培训计划，并在管理上提供包括经费在内的支持。

（2）制订培训目标和具体的培训计划。

（3）成立培训组，配备相应的工具、设备和教材。

3）软件全生命周期测试

提高测试成熟度和改善软件产品质量都要求将测试工作与软件生命周期中的各个阶段联系起来。该目标有 4 个子目标：

（1）将测试阶段划分为子阶段，并与软件生命周期的各阶段相联系。

（2）基于已定义的测试子阶段，采用软件生命周期 V 字模型。

（3）制订与测试相关的工作产品的标准。

（4）建立测试人员与开发人员共同工作的机制。这种机制有利于促进将测试活动集成于软件生命周期中。

4）控制和监视测试过程

软件组织采取如下措施：制订测试产品的标准，制订与测试相关的偶发事件的处理预案，确定测试里程碑，确定评估测试效率的度量，建立测试日志等。控制和监视测试过程有 3 个子目标：

（1）制订控制和监视测试过程的机制和政策。

（2）定义、记录并分配一组与测试过程相关的基本测量。

（3）开发，记录并文档化一组纠偏措施和偶发事件处理预案，以备实际测试严重偏离计划时使用。

在 TMM 的定义级，测试过程中引入计划能力，在 TMM 的集成级，测试过程引入控制和监视活动。两者均为测试过程提供了可见性，为测试过程持续进行提供保证。

4. 管理和测量级

TMM 的管理和测量级中，测试活动包括软件生命周期中各个阶段的评审、审查和追查，使得测试活动涵盖软件验证和确认活动。因为测试是可以量化并度量的过程，根据管理和测量级要求，与软件测试相关的活动，如测试计划、测试设计和测试步骤都要经过评审。为了测量测试过程，建立测试数据库，用于收集和记录测试用例，记录缺陷并按缺陷

的严重程度划分等级。此外，所建立的测试规程应能够支持软件组中对测试过程的控制和测量。

管理和测量级有3个要实现的成熟度目标：建立组织范围内的评审程序，建立测试过程的测量程序和软件质量评价。

1）建立组织范围内的评审程序

软件组织应在软件生命周期的各阶段实施评审，以便尽早有效地识别，分类和消除软件中的缺陷。建立评审程序有4个子目标：

（1）管理层要制订评审政策支持评审过程。

（2）测试组和软件质量保证组要确定并文档化整个软件生命周期中的评审目标，评审计划，评审步骤以及评审记录机制。

（3）评审项由上层组织指定。通过培训参加评审的人员，使他们理解和遵循相关的评审政策、评审步骤。

2）建立测试过程的测量程序

测试过程的测量程序是评价测试过程质量，改进测试过程的基础，对监视和控制测试过程至关重要。测量包括测试进展，测试费用，软件错误和缺陷数据以及产品测量等。建立测试测量程序有3个子目标：

（1）定义组织范围内的测试过程测量政策和目标。

（2）制订测试过程测量计划。测量计划中应给出收集、分析和应用测量数据的方法。

（3）应用测量结果制订测试过程改进计划。

3）软件质量评价

软件质量评价内容包括定义可测量的软件质量属性，定义评价软件工作产品的质量目标等项工作。软件质量评价有两个子目标：

（1）管理层，测试组和软件质量保证组要制订与质量有关的政策，质量目标和软件产品质量属性。

（2）测试过程应是结构化，已测量和已评价的，以保证达到质量目标。

5. 优化，预防缺陷和质量控制级

本级的测试过程是可重复、可定义、可管理，因此软件组织优化调整和持续改进测试过程。测试过程的管理为持续改进产品质量和过程质量提供指导，并提供必要的基础设施。

优化，预防缺陷和质量控制级有3个要实现的成熟度目标：

1）应用过程数据预防缺陷

此时的软件组织能够记录软件缺陷，分析缺陷模式，识别错误根源，制订防止缺陷再次发生的计划，提供跟踪这种活动的办法，并将这些活动贯穿于全组织的各个项目中。应用过程数据预防缺陷的成熟度子目标：

（1）成立缺陷预防组。

（2）识别和记录在软件生命周期各阶段引入的软件缺陷和消除的缺陷。

（3）建立缺陷原因分析机制，确定缺陷原因。

（4）管理，开发和测试人员互相配合制订缺陷预防计划，防止已识别的缺陷再次发生。缺陷预防计划要具有可跟踪性。

2）质量控制在本级

软件组织通过采用统计采样技术，测量组织的自信度，测量用户对组织的信赖度以及设定软件可靠性目标来推进测试过程。为了加强软件质量控制，测试组和质量保证组要有负责质量的人员参加，他们应掌握能减少软件缺陷和改进软件质量的技术和工具。支持统计质量控制的子目标有：

（1）软件测试组和软件质量保证组建立软件产品的质量目标，如产品的缺陷密度、组织的自信度以及可信赖度等。

（2）测试管理者要将这些质量目标纳入测试计划中。

（3）培训测试组学习和使用统计学方法。

（4）收集用户需求以建立使用模型。

3）优化测试过程在测试成熟度的最高级

已能够量化测试过程。这样就可以依据量化结果来调整测试过程，不断提高测试过程能力，并且软件组织具有支持这种能力持续增长的基础设施。基础设施包括政策、标准、培训、设备、工具以及组织结构等。优化测试过程包含：

（1）识别需要改进的测试活动。

（2）实施改进。

（3）跟踪改进进程。

（4）不断评估所采用的与测试相关的新工具和新方法。

（5）支持技术更新。

4）测试过程优化所需子成熟度目标

（1）建立测试过程改进组，监视测试过程并识别其需要改进的部分。

（2）建立适当的机制以评估改进测试过程能力和测试成熟度的新工具和新技术。

（3）持续评估测试过程的有效性，确定测试终止准则。终止测试的准则要与质量目标相联系。

总之，TMM 5 个阶段总结如下：第一阶段：测试和调试没有区别，除了支持调试外，测试没有其他目的。第二阶段：测试的目的是为了表明软件能够工作。第三阶段：测试的目的是为了表明软件能够正常工作。第四阶段：测试的目的不是要证明什么，而是为了把软件不能正常工作的预知风险降低到能够接受的程度。第五阶段：测试成为了自觉的约束，只需少量的测试投入便可产生高质量、低风险的软件。综上所述，表 10.1 给出了测试成熟度模型的基本描述。

至此，TMM 的特征如下：

（1）TMM 从 CMM 延伸到测试，容易脱离测试过程的改进轨迹。例如，许多项目测试是从缺陷开始，将缺陷划分为不同等级，先有测试用例，才有测试计划。

（2）关注测试自身活动不够。TMM 中包括测试工具和测试数据库，但几乎没有涉及缺陷清除率、自动化测试框架和测试流程等。

表 10.1　测试成熟度模型的基本描述

级别	简 单 描 述	特　　征	目　　标
初始级	测试处于一个混乱的状态，缺乏成熟的测试目标，测试处于可有可无的地位	还不能把测试同调试分开；编码完成后才进行测试工作；测试的目的是表明程序没有错；缺乏相应的测试资源	——
定义级	测试目标是验证软件符合需求，会采用基本的测试技术和方法	测试被看做是有计划的活动；测试同调试分开；在编码完成后才进行测试工作	启动测试计划过程；将基本的测试技术和方法制度化
集成级	测试不再是编码后的一个阶段，而是贯穿在整个软件生命周期中，测试建立在满足用户或客户的需求上	具有独立的测试部门；根据用户需求设计测试用例；有测试工具辅助进行测试工作；没有建立起有效的评审制度；没有建立起质量控制和质量度量标准	建立软件测试组织；制订技术培训计划；测试在整个生命周期内进行；控制和监视测试过程
管理和度量级	测试是一个度量和质量的控制过程。在软件生命周期中评审被作为测试和软件质量控制的一部分	进行可靠性、可用性和可维护性等方面的测试；采用数据库来管理测试用例；具有缺陷管理系统并划分缺陷的级别；还没有建立起缺陷预防机制，缺乏自动对测试中产生的数据进行收集和分析的手段	实施软件生命周期中各阶段评审；建立测试数据库并记录、收集有关测试数据；建立组织范围内的评审程序；建立测试过程的度量方法和程序；进行软件质量评价
优化级	具有缺陷预防和质量控制的能力，已经建立起测试规范和流程，并不断地进行测试改进	运用缺陷预防和质量控制措施；选择和评估测试工具存在一个既定的流程；测试自动化程度高；自动收集缺陷信息；有常规的缺陷分析机制	应用过程数据预防缺陷，统计质量控制，建立软件产品的质量目标，持续改进、优化测试过程

（3）测试深度不够。测试具有很大风险因素，如何降低测试风险，提高测试用例的质量，TMM 并没有给出清晰的目标。

（4）在过程监控、度量方面的内容较多，但是如何对实际测试能力（测试覆盖率、效率等）进行量化评估，TMM 并没有具体的对策，实施性较差。

10.3　测试工具原理

白盒测试工具、黑盒测试工具、测试设计与开发工具、测试执行和评估工具以及测试管理工具的测试原理各不相同，下面依次进行介绍。

10.3.1　白盒测试工具

白盒测试工具应用在具有高可靠性的软件领域，例如军工软件、航天航空软件、工业控制软件等。白盒测试工具是对源代码进行测试，主要测试词法分析与语法分析、静态错误分析、动态检测等。目前白盒测试工具主要支持标准 C、C++、Visual C++、Java、Visual J++ 等程序开发语言。

根据测试工具原理不同，白盒测试又分为静态测试工具和动态测试工具。静态测试工具直接对代码进行分析，不需要运行代码，也不需要对代码编译链接和生成可执行文件。静态测试工具一般是对代码进行语法扫描，找出不符合编码规范的地方，根据某种质量模型评价代码的质量，生成系统的调用关系图等。静态测试工具的代表有 Telelogic 公司的 Logiscope 软件、PR 公司的 PRQA 软件。

按照完成职能的不同，静态测试工具又有以下几种类型：

(1) 代码覆盖率分析器和代码测量器。代码分析类似于高级编译系统，一般针对高级语言构造分析工具，定义类、对象、函数、变量等定义规则、语法规则，对代码进行语法扫描、跟踪程序逻辑，观看程序的图形表达，找出不符合编码规范的地方，确认死代码，根据某种质量模型评价代码质量、生成系统的调用关系图等。此类工具能够量化设计的复杂度，限制测试所必需的集成测试的数量，有助于进行集成测试。此外，工具还能用多种方式测量测试覆盖率，其中包括代码段、分支段和条件值覆盖率，有助于把没有覆盖到的分支和逻辑结构加入到测试集中。

(2) 一致性检查。一致性检查检测程序的各个单元是否使用了统一的记法和术语，用于检查是否遵循了设计规格说明，称为一致性检查器。

(3) 接口分析。接口分析检查程序单元之间接口的一致性，以及是否遵循预先确定的规则和原则。典型的接口分析包括检查传送给子程序的参数以及检查模块的完整性，称为接口检查器。

(4) 类型分析。类型分析检测数据的赋值与引用之间是否出现了不合理的现象，如引用了未赋予的变量，对以前未曾引用的变量的再次赋值等数据流异常现象。

动态测试工具一般采用“插桩”的方式，插入一些监测代码，用来统计程序运行时的数据。动态测试工具的代表有 Compuware 公司 DevPartner 软件、Rational 公司 Purify 系列。按照完成职能的不同，动态测试工具分为以下几种类型：

(1) 功能确认与接口测试。功能确认与接口测试用于测试各个模块功能、模块之间的接口、局部数据结构、主要执行路径、错误处理等。

(2) 性能与内存分析。负载/性能测试工具模拟系统的真实负载，检查系统或者应用程序的响应时间和负载能力。强度测试工具模拟高强度场景运行来确定软件是否会崩溃和什么时候崩溃。性能分析用于查找为修改程序的瓶颈，从而改变整个系统性能。此类工具用于验证应用程序是否正确地使用它的内存资源，确定应用程序是否释放了它所申请的内存，并且提供运行时的错误检测。因为许多程序缺陷都和内存问题有关，其中包括性能问题，如果应用程序对于内存的操作非常频繁，进行内存检测是非常必要的。通过分析内存使用情况，了解程序内存分配的真实情况，发现内存的不正常使用，在问题出现前发现征兆，发现内存分配错误，找出发生故障的原因。

10.3.2 黑盒测试工具

黑盒测试工具是指测试软件功能或性能的工具，主要用于系统测试和验收测试，检测每个功能是否都能按照需求规格说明的规定正常工作。黑盒测试工具有 Rational 公司的 TeamTest、Robot；Compuware 公司的 QACenter，MI 公司的 WinRunner 和 Quick

Test Professional 等。黑盒测试工具一般具有如下功能：

1. 虚拟用户技术

虚拟用户技术通过模拟真实用户行为对被测程序(Application UnderTest，AUT)施加负载，测量 AUT 的性能指标值，如事务的响应时间、服务器吞吐量等。虚拟用户技术以真实用户的"商务处理"(用户为完成一个商业业务而执行的一系列操作)作为负载的基本组成单位，用"虚拟用户"(模拟用户行为的测试脚本)模拟真实用户。

负载需求(例如并发虚拟用户数、处理的执行频率等)通过人工收集和分析系统使用信息来获得，负载测试工具模拟成千上万个虚拟用户同时访问 AUT，来自不同 IP 地址、不同浏览器类型以及不同网络连接方式的请求，并实时监视系统性能，帮助测试人员分析测试结果。虚拟用户技术具有成熟测试工具支持，但确定负载的信息要依靠人工收集，准确性不高。

2. 脚本技术

脚本是一组测试工具执行的指令集合，也是计算机程序的另一种形式表现形式。脚本语言至少具有如下的功能：

- 支持多种常用的变量和数据类型。
- 支持各种条件逻辑、循环结构。
- 支持函数的创建和调用。

脚本有两种，一种是手动编写或嵌入源代码；另一种是通过测试工具提供的录制功能，运行程序自动录制生成脚本。由于录制生成脚本过于简单，仅靠自动录制脚本，无法满足用户复杂要求，需要添加参数设置，增强脚本的实用性。

脚本技术分为以下几种类型：

1) 线性脚本

录制手工执行的测试用例得到的线性脚本，包含用户键盘和鼠标输入，检查某个窗口是否弹出等操作。

线性脚本具有如下一些优点：不需要深入的工作或计划，对实际执行操作可以审计跟踪。线性脚本适用于演示、培训或执行较少且环境变化小的测试、数据转换的操作功能。但是，线性脚本具有以下缺点：过程较繁琐，过多依赖于每次捕获内容，测试输入和比较是"捆绑"在脚本中，不能共享或重用脚本，容易受软件变化的影响。另外，线性脚本修改代价大，维护成本高，容易受意外事件影响，导致整个测试失败。

2) 结构化脚本

结构化脚本类似于结构化程序设计，包含控制脚本执行指令，具有顺序、循环和分支等结构。结构化脚本的优点是健壮性好，通过循环和调用减少工作量；但结构化脚本比较复杂，而且测试数据仍然与脚本"捆绑"在一起。

3) 共享脚本

共享脚本侧重描述脚本中共享的特性，脚本可以被多个测试用例使用，一个脚本可以被另一个脚本调用。当重复任务发生变化，只需修改一个脚本，便可达到脚本共享目的。

共享脚本的优点：以较少的开销实现类似的测试，维护共享脚本的开销低于线性脚本。但是，共享脚本需要跟踪更多的脚本，给配置管理带来一定困难，并且对于每个测试用例仍然需要特定的测试脚本。

4）数据驱动脚本

数据驱动脚本将测试输入到独立的数据文件（数据库）中，而不是绑定在脚本中。执行时是从数据文件中读数据，使得同一个脚本执行不同的测试，只需对数据进行修改，不必修改执行脚本。通过一个测试脚本指定不同的测试数据文件，实现较多的测试用例，将数据文件单独列出，选择合适的数据格式和形式，达到简化数据、减少出错的目的。

数据驱动脚本具有如下优点：快速增加类似的测试用例，新增加的测试也不必掌握工具脚本技术，对以后类似的测试无需额外的维护，有利于测试脚本和输入数据分离，减少编程和维护的工作量，有利于测试用例扩充和完善。但是，数据驱动脚本初始建立开销较大、需要专业人员支持。

5）关键字驱动脚本

关键字驱动作为比较复杂的数据驱动技术的逻辑扩展，是将数据文件变成测试用例的描述，用一系列关键字指定要执行的任务。关键字驱动技术假设测试者具有被测系统知识和技术，不必告之如何进行详细动作，以及测试用例如何执行，只说明测试用例即可。关键字驱动脚本多使用说明性方法和描述性方法。

10.3.3 测试设计和开发工具

测试设计是说明测试被测试软件特征或特征组合的方法，确定并选择相关测试用例的过程。常用的测试数据生成工具有：Bender & Associates 公司提供的功能测试数据生成工具 SoftTest，Parasoft 公司提供的 Jtest（用于 Java 的代码分析和动态类、组件测试）、Jcontract（用于 Java 的实时性能监控以及分析优化）、C++ Test（用于 C 和 C++ 语言的代码分析和动态测试）、CodeWizard（用于 C 和 C++ 的代码静态分析）、Insure ++（用于 C 和 C++ 的实时性能监控以及分析优化）。

测试设计与开发需要的工具类型有：

1. 测试数据生成器

测试数据生成器通过自动生成测试数据来辅助测试过程。目前，市场上有多种工具支持生成测试数据。无论测试数据是用于功能测试、数据驱动的负载测试，还是性能测试和强度测试，测试数据生成器都能够根据一组规则快速地生成测试数据库。

2. 录制/回放

录制/回放是黑盒测试的自动化方法，通过捕获用户每一步操作，如用户界面的像素坐标或程序显示对象（窗口、按钮、滚动条等）的位置，以及相应操作、状态变化或属性变化，用一种脚本语言记录描述，模拟用户操作。回放时，将脚本语言转换为屏幕操作，比较被测系统的输出记录与预先给定的标准结果。

目前的自动化负载测试解决方案几乎都是采用“录制-回放”的技术。所谓的“录制-

回放”技术，就是先由手工完成一遍测试流程，由计算机记录下这个流程期间客户端和服务器端之间的通信信息，这些信息通常是一些协议和数据，并形成特定的脚本程序。然后在系统的统一管理下同时生成多个虚拟用户，并运行该脚本，监控硬件和软件平台的性能，提供分析报告或相关资料。通过模拟出成百上千的用户对应用系统进行负载能力的测试。

10.3.4 测试执行和评估工具

测试执行和评估是执行测试用例并对测试结果进行评估的过程，包括选择用于执行的测试用例、设置测试环境、运行所选择的测试用例、记录测试执行过程、分析潜在的软件故障并测量测试工作的有效性。评估类工具对执行测试用例和评估测试结果起辅助作用。

测试执行和评估工具有：

（1）度量报告工具：用于读取源代码并显示度量信息，如数据流、数据结构和控制流的复杂度，根据模块、操作数、操作符和代码的数量提供代码规模的度量分析数据。

（2）存储器测试工具：用于检测存储器初始化与读取等问题，检测对于已分配但未释放内存。

10.3.5 测试管理工具

测试管理工具是指管理整个测试流程的工具，用于测试计划的管理、测试用例的管理、缺陷跟踪、测试报告管理等，一般贯穿于整个软件测试生命周期。测试管理工具有 Rational 公司的 Testmanager 、ClearQuest 等，Compuware 公司的 QACenter 和 TrackRecord 等。

下面介绍主要的测试管理工具。

1. 测试过程生成器

需求管理工具与基于需求说明书的测试过程生成器联成一体，当需求管理工具捕捉到需求信息后，这些信息会被测试过程生成器利用，生成器通过统计、计算或者探索式的方法创建测试过程。若使用统计的方法生成测试过程，工具会按一个分布选择输入值，这个分布可能是统计上的随机分布，或者是和正在测试的软件的用户配置相匹配的分布。测试数据生成器最常使用的策略是动作、数据、逻辑、事件和状态驱动，这些策略用于检测不同种类的软件缺陷。

2. 测试用例管理

测试用例管理具有如下一些功能：

- 提供用户界面用于管理测试。
- 对测试进行整理，方便使用和维护。
- 启动并管理测试执行，运行用户选择的测试。
- 提供与捕获/回放及覆盖分析工具的集成。
- 提供自动化的测试报告和相关文档的编制。

3. 缺陷跟踪管理

缺陷跟踪管理又称为问题跟踪工具、故障管理工具等，用于在整个软件生命周期中对缺陷进行跟踪管理和强化管理记录、跟踪并提供全面的帮助。缺陷跟踪管理具有如下一些特征：

- 迅速提交和更新故障报告。
- 有选择地自动通知用户对故障状态进行修改。
- 具有对数据的安全访问。

4. 配置管理

配置管理的目标就是为了标识变更、控制变更、确保变更正确实现并向其他有关人员报告变更。从某种角度讲，配置管理是一种标识、组织和控制修改的技术，目的是使错误降为最小并最有效地提高生产效率。

10.4 测试工具选择

自动化测试工具具有如下特征。

1. 支持脚本语言、函数库

支持脚本语言的函数库作为测试工具的最基本要求。程序即使作了修改，只需要把原脚本中的相应函数进行更改，而不用改动所有可能的脚本，节省大量工作。另外，通过对外部函数的支持，如对 DLL 文件的访问，对数据库编程接口的调用，获得强大的功能。

2. 对程序界面中对象的识别能力

测试工具必须能够将程序界面中的相应的对象（如按钮、文本框、表单等）区分并识别，录制的测试脚本才能具有良好的可读性、修改的灵活性和维护的方便性。如果只是简单通过像素位置坐标区分对象，就会存在较多问题，例如界面稍微改变、或者屏幕的分辨率、测试环境的改变，会导致原有的测试脚本无法使用。

3. 抽象层

抽象层和对象识别能力有一定的关系。用于捕捉回放程序界面过程中，抽象层一般位于被测应用程序和录制生成的测试脚本之间，抽象层用于将程序界面中存在对象实体映射成逻辑对象，测试针对逻辑对象进行，不需依赖界面的对象实体，减少测试脚本建立和维护的工作量。

4. 分布式测试的网络支持

互联网软件，如网络会议系统、远程培训系统、聊天系统等软件，一般都具有协同工作、相互通信等模式，支持多用户共同操作，这些软件的测试有如下要求：

（1）测试工具在进行测试时传输的数据量要小，要具有独立性，避免受到被测软件的影响。

（2）按照设置的任务执行时间表进行，即在指定时间执行指定的测试任务。

（3）当两个测试任务并发时，需要能保持协调或协同处理，避免出现资源竞争问题。

5. 图表功能

测试工具具有将测试结果生成一些统计报表，利于测试人员的工作。

6. 测试工具的集成能力

测试工具的引入是一个长期的过程，伴随着测试过程改进的一个持续的过程。因此，测试工具应于开发工具进行良好的集成，并且也能够和其他测试工具的集成。

当前市场上测试工具很多，每个测试工具在不同环境由其各自的优点和缺点。如何选择最佳的测试工具，主要依赖于系统工程环境以及组织特定的其他需求和标准。因此，选择自动化测试工具应从以下方面考虑：

（1）测试工具的集成能力。确认测试工具与操作系统、编程语言环境和其他方面相兼容。确定各种系统构架。必须确定应用程序在技术上的构架，其中包括整个组织或者项目使用的中间件、数据库、操作系统、开发语言、使用的第三方插件等。

（2）确定被测试应用程序管理数据的方式。必须了解被测试应用程序管理数据的方式，确定自动测试工具如何支持对数据的验证。

（3）确定测试类型。必须了解工具的测试类型，如用于回归测试、强度测试或者容量测试等测试工具功能差距较大。

（4）确定项目进度。测试工具是否影响测试进度，在进度时间表内，评审测试人员是否有足够的时间学习使用测试工具非常重要。

（5）确定项目预算。通过分析成本与效益，确定所投入的总成本与获益之间的关系。

10.5 习　　题

1. 选择题

（1）使用软件测试工具的目的________。

A. 帮助测试寻找问题　　B. 协助问题的诊断

C. 节省测试时间　　D. 提高 Bug 的发现率

E. 更好的控制缺陷提高软件质量　　F. 更好的协助开发人员

（2）RATIONAL 软件包中，用于测试企业互联网网页站点的工具为________。

A. Rational Administrator　　B. Rational Test Manager

C. Rational Site Check　　D. Rational Rational TestFactory

(3) 在 RATIONAL 的脚本创建过程中，能为脚本回放期间提供数据值的命令为________。

A. Query B. Standard C. Datapool D. Available

(4) 软件性能测试工具有________。

A. loadRunner B. Rational Visual Quantify

C. PureLoad D. 以上都是

2. 简答题

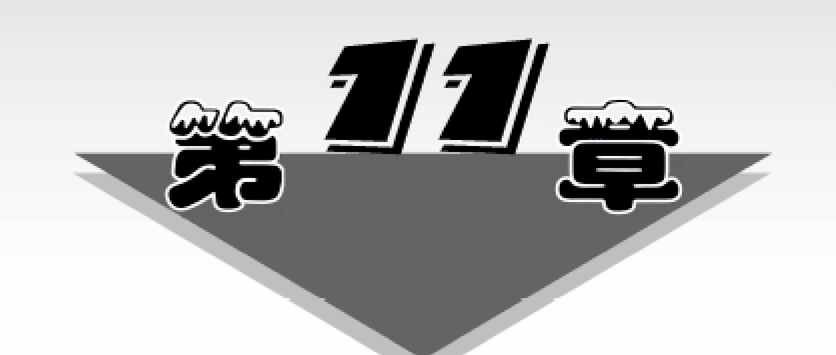
(1) 自动化测试相对于手工测试有什么好处?

(2) 自动测试发展经过了几个阶段?

(3) 测试成熟度模型是什么? 包括哪些内容?

(4) 录制和回放是指什么?

(5) 脚本技术可以分为几类? 分别是什么?



测试管理工具

11.1 概　　述

测试管理工具是对测试计划、测试用例、测试实施、缺陷跟踪进行管理,用于提高测试效率、提升测试质量、测试用例复用率等。目前市场上主流的软件测试管理工具有Rational 公司的 Test Manager、Compureware 公司的 TrackRecord、Mercury Interactive 公司的 TestDirector 以及 TestCenter 等软件。

表 11.1 给出了 TestManager、Wiki、Bugzilla、TestDirector 等软件的各自内容。

表 11.1　各种测试用例管理工具

工具名	优　　点	缺　　点
TestManager	1. 功能强大 2. 对测试用例无限分级 3. 可以和 Rational 的测试工具 robot、functional 相结合 4. 有测试用例执行的功能,但必须先生成对应的手工或自动化脚本	1. 本地化支持不好,汉字显示太小 2. 测试用例不太稳定 3. 必须安装客户端才可使用,和开发人员交流不方便 4. 测试用例的展示形式单一
Wiki	1. 具有 Web 界面形式,交流方便 2. 测试用例形式多样 3. Wiki 提供测试用例的版本控制与版本比较功能 4. Wiki 提供测试用例加注释功能,方便测试用例评审 5. Wiki 本身强大的全文索引功能 6. 可以任意为测试用例添加标签	1. 并不是专业的测试用例管理工具 2. 无法和其他测试工具集成 3. 测试用例的统计不方便 4. 没有测试用例的执行跟踪功能 5. 有一些 Wiki 本身的限制,如不同产品的测试用例名也不能重复 6. 目前还没有定制统一的模板
Bugzilla	1. 开源免费 2. Web 方式的管理界面 3. 自动邮件提醒 4. 与 Bugzilla 结合紧密 5. 测试用例可以分优先级 6. 测试用例可以有评审的功能	1. 安装设置较繁琐 2. 编写测试用例必须按照一个步骤对应一个验证点的形式来编写
TestDirector	1. 可以和 Rational 测试工具结合 2. 具有 Web 方式的界面 3. 有测试用例执行跟踪的功能	1. 每个项目库同时在线人数有限制 2. 存在不稳定性

续表

工具名	优　　点	缺　　点
TestDirector	4. 有灵活的缺陷定制 5. 和自身的缺陷管理工具紧密集成 6. 界面较友好	
TestLink	1. 具有 Web 方式的界面 2. 可以和 Bugzilla 等缺陷管理工具整合 3. 同时具有需求管理的功能	
Excel 形式	功能强大	维护较麻烦，统计、度量等也不方便
Word 形式	很灵活，易于扩展	不如 Excel 格式统一，也不如 Excel 容易统计

11.2　测试管理工具——TestDirector

11.2.1　TestDirector 简介

TestDirector 是 HP 公司推出的基于 Web 的测试管理工具，能够系统地控制整个测试过程，并创建整个测试工作流的框架和基础，使整个测试管理过程变得更为简单和有组织。TestDirector 通过工程数据库，将每一个测试点都对应着一个指定的测试需求，并提供直观和有效的方式来计划和执行测试集、收集测试结果并分析数据。

TestDirector 提供完善的缺陷跟踪系统，能够跟踪缺陷从产生到最终解决的全过程。TestDirector 与 WinRunner、LoadRunner 等需求和配置管理工具、建模工具无缝链接，提供可供选择的全套解决方案，用于进行全部自动化的应用测试。

TestDirector 的测试管理包括需求定义、测试计划、测试执行和缺陷跟踪四个阶段。

1. 需求定义

需求分析用于分析应用程序并确定测试需求。具体内容如下：

（1）定义测试范围：检查应用程序文档，并确定测试范围——测试目的、目标和策略。

（2）创建需求：创建需求树，并确定它是否涵盖所有的测试需求。

（3）描述需求：为“需求树”中的每一个需求主题建立了一个详细的目录，并描述每一个需求，给它分配一个优先级，如有必要的话还可以加上附件。

（4）分析需求：产生报告和图表来帮助分析测试需求，并检查需求以确保它们在测试范围内。

2. 测试计划

测试计划基于测试需求，建立测试计划。具体内容如下：

（1）定义测试策略：检查应用程序、系统环境和测试资源，并确认测试目标。

（2）定义测试主题：将应用程序基于模块和功能进行划分，并对应到各个测试单元

或主题，构建测试计划树。

（3）定义测试：定义每个模块的测试类型，并为每一个测试添加基本的说明。

（4）创建需求覆盖：将每一个测试与测试需求进行连接。

（5）设计测试步骤：对于每一个测试，先决定其要进行的测试类型，若准备进行手动测试，需要为其在测试计划树上添加相应的测试步骤。测试步骤描述测试的详细操作、检查点和每个测试的预期结果。

（6）分析测试计划：产生报告和图表来帮助分析测试计划数据，并检查所有测试以确保它们满足测试目标。

3. 测试执行

测试执行创建测试集并执行测试。具体内容如下：

（1）创建测试集：在工程中定义不同的测试组来达到各种不同的测试目标，并确定每个测试集都包括了哪些测试。

（2）确定进度表：为测试执行制定时间表，并为测试员分配任务。

（3）运行测试：自动或手动执行每一个测试集。

（4）分析测试结果：查看测试结果并确保应用程序缺陷已经被发现。生成的报告和图表可以帮助分析这些结果。

4. 缺陷跟踪

缺陷跟踪报告程序中产生的缺陷并跟踪缺陷修复的全过程。具体内容如下：

（1）添加缺陷：报告程序测试中发现的新的缺陷。在测试过程中的任何阶段，质量保证人员、开发者、项目经理和最终用户都能添加缺陷。

（2）检查新缺陷：检查新的缺陷，并确定哪些缺陷应该被修复。

（3）修复打开的缺陷：修复那些决定要修复的缺陷。

（4）测试新构建：测试应用程序的新构建，重复上面的过程，直到缺陷被修复。

（5）分析缺陷数据：产生报告和图表来帮助分析缺陷修复过程，并帮助决定什么时候发布该产品。

11.2.2 TestDirector 使用概述

1. 启动 TestDirector

用户在“开始”菜单单击 TestDirector 图标，启动 TestDirector 后，首页如图 11.1 所示，地址栏为：http://服务器名(或者 IP 地址)/TDBIN。

单击左侧的 TestDirector，进入 TestDirector 的主界面，选择域和工程，输入用户名和密码，登录 TestDirector 主界面。本书选用默认的域和工程，采用自带的 Demo 项目例子，用默认超级用户 Admin，输入密码(默认为空)，单击 Login，登录 TestDirector。

单击 TestDirector 首页左侧的 Site Administrator，输入密码(默认为空)，进入 TestDirector 管理界面，如图 11.2 所示。



图 11.1　TestDirector 登录界面

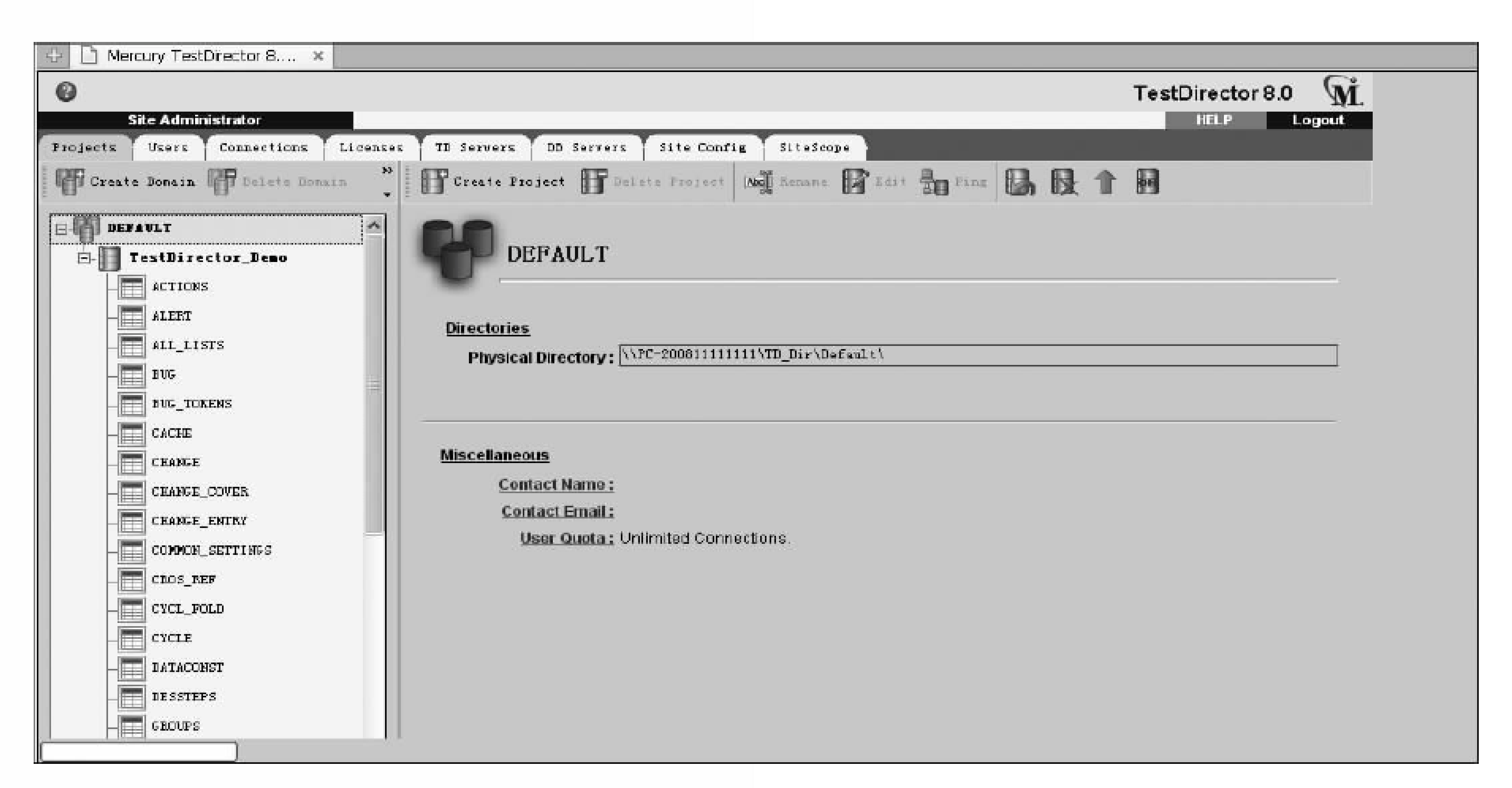


图 11.2　TestDirector Site Administrator 界面

2. TestDirector 主窗口

TestDirector 的主窗口由以下几部分构成，如图 11.3 所示。

- TestDirector Toolbar：显示工程常用的命令。
- Menu Bar：显示模块常用的命令。
- Tools Button：显示常用的工具。

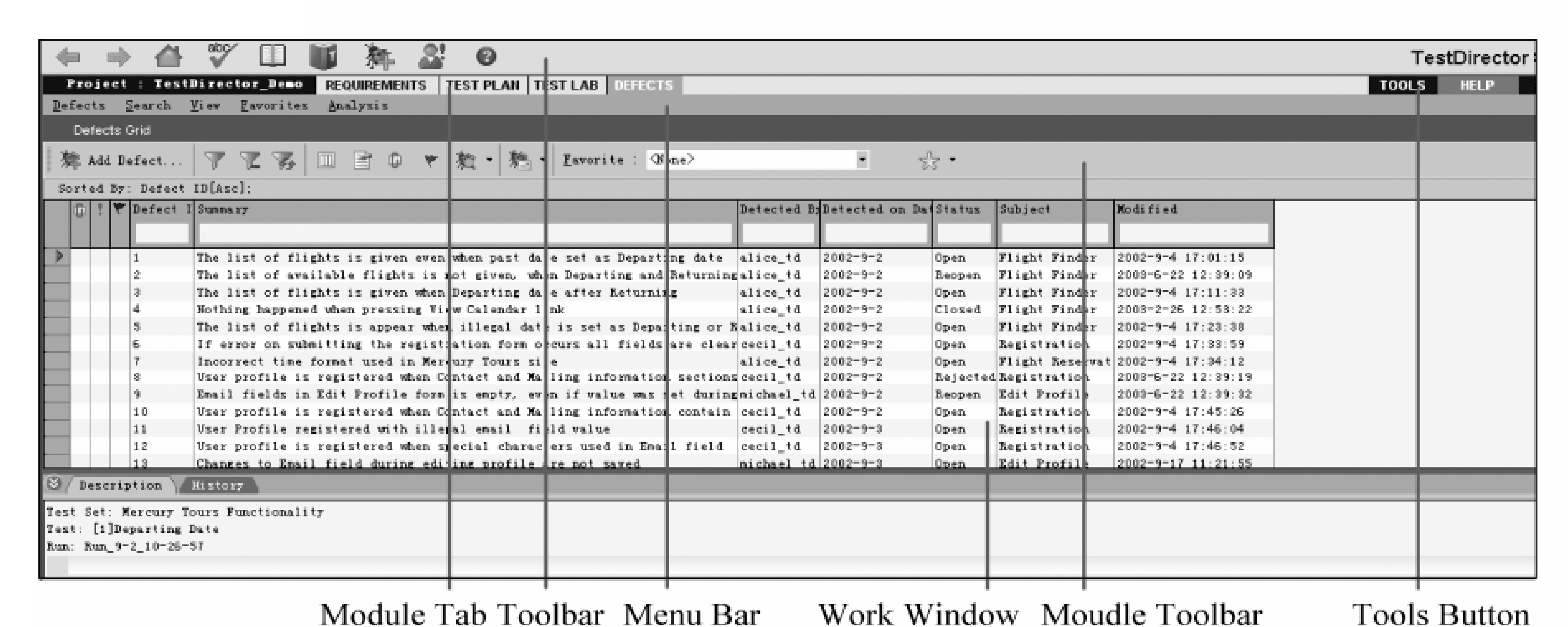


图 11.3 TestDirector 主窗口

3. TestDirector 应用举例

1）项目管理库环境配置

登录 Site Administrator 后，在 Projects 标签页中创建域，只有系统的域用户可以共享这个域的存储区，一个域可以存储多个工程。

当创建一个 TestDirector 工程后，需要存储和管理 TestDirector 自身产生和连接的数据库。每一个工程都支持通过数据库来存储工程信息。在默认域(default)中创建测试项目 tdtest，数据库类型为 Microsoft SQL。在工具栏选择 Creat Project，输入项目名称，选择数据库 MS-SQL，如图 11.4 所示。

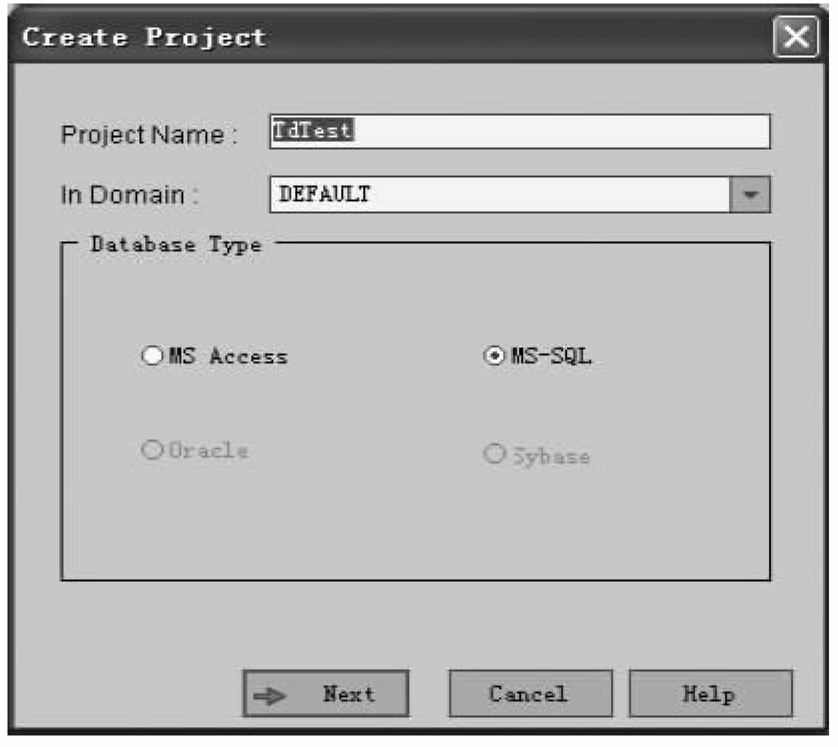


图 11.4 TestDirector 创建测试项目

单击 Next 按钮，输入数据库网络服务名，数据库用户名和密码(要具有创建数据库表的权限)，连接成功后，给出提示信息如图 11.5 和图 11.6 所示。

单击 Creat 按钮，系统创建数据库及相应表，创建成功后，数据库中自动生成用户：域名_测试项目名_db，默认密码为空，对于本例，用户为：default_tdtest_db，密码为空，如图 11.7 所示。

TestDirector 允许管理用户访问工程的权限，它会创建一个有权用户的列表和为一个组或者是一个用户分配一个口令。在 TestDirector 中用户所拥有的权利是由该用户所在的用户组决定的。TestDirector 允许为工程中指定的目录创建包含特权和许可机制的规则。

系统中添加用户，单击 Users 标签，选择 New 添加用户，输入新增用户信息，单击 OK 按钮后，成功加入该用户，如图 11.8 所示。

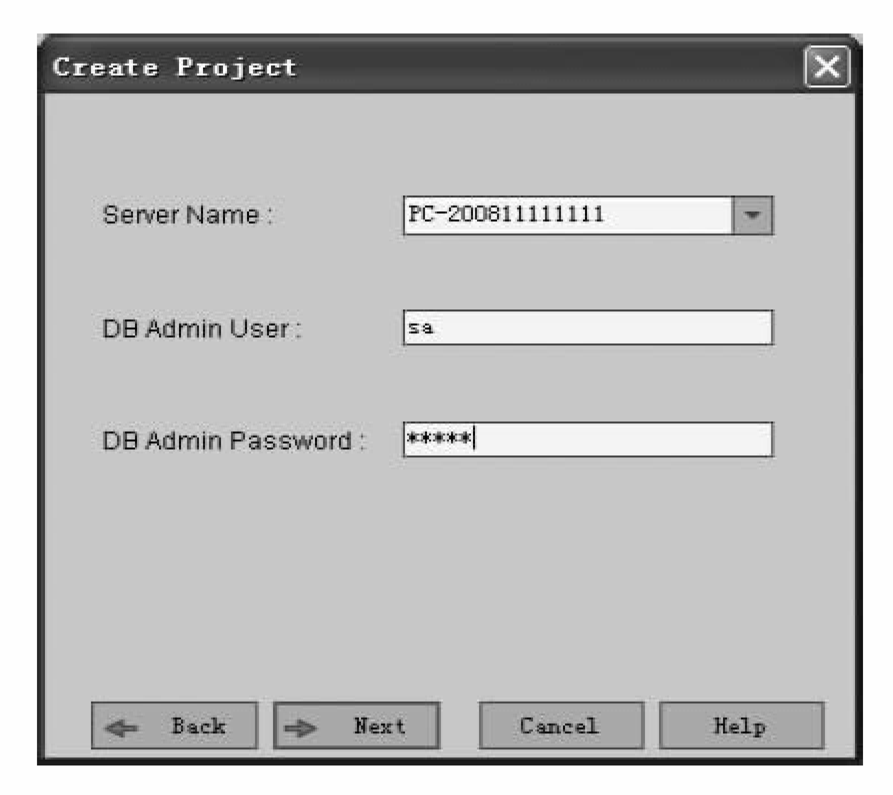


图 11.5　TestDirector 连接数据库

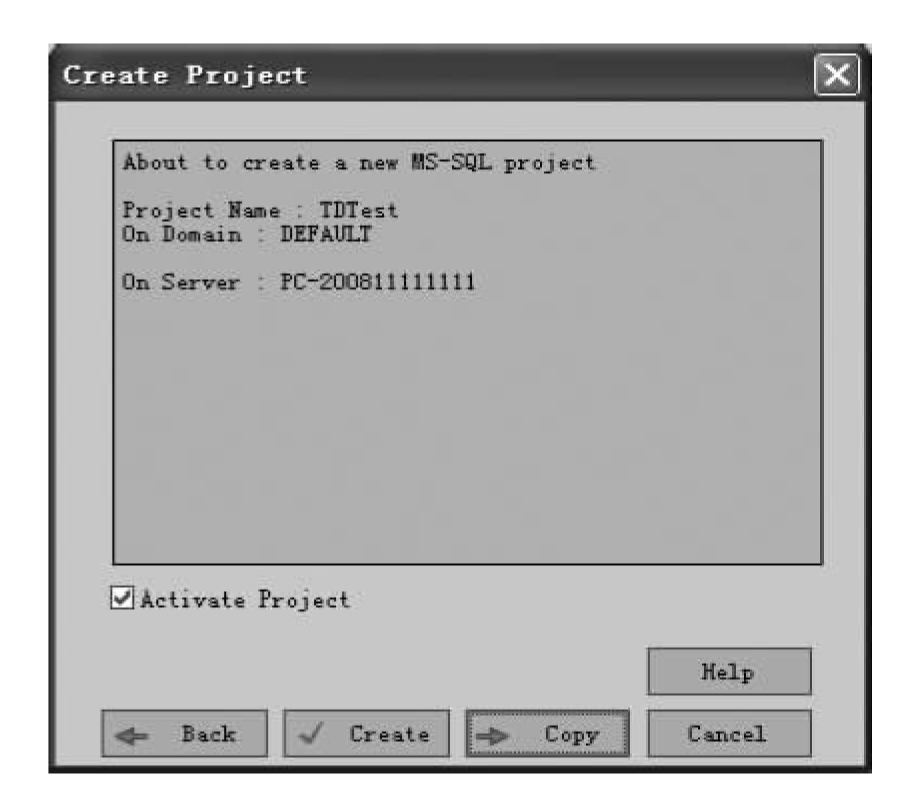


图 11.6　TestDirector 创建数据库信息

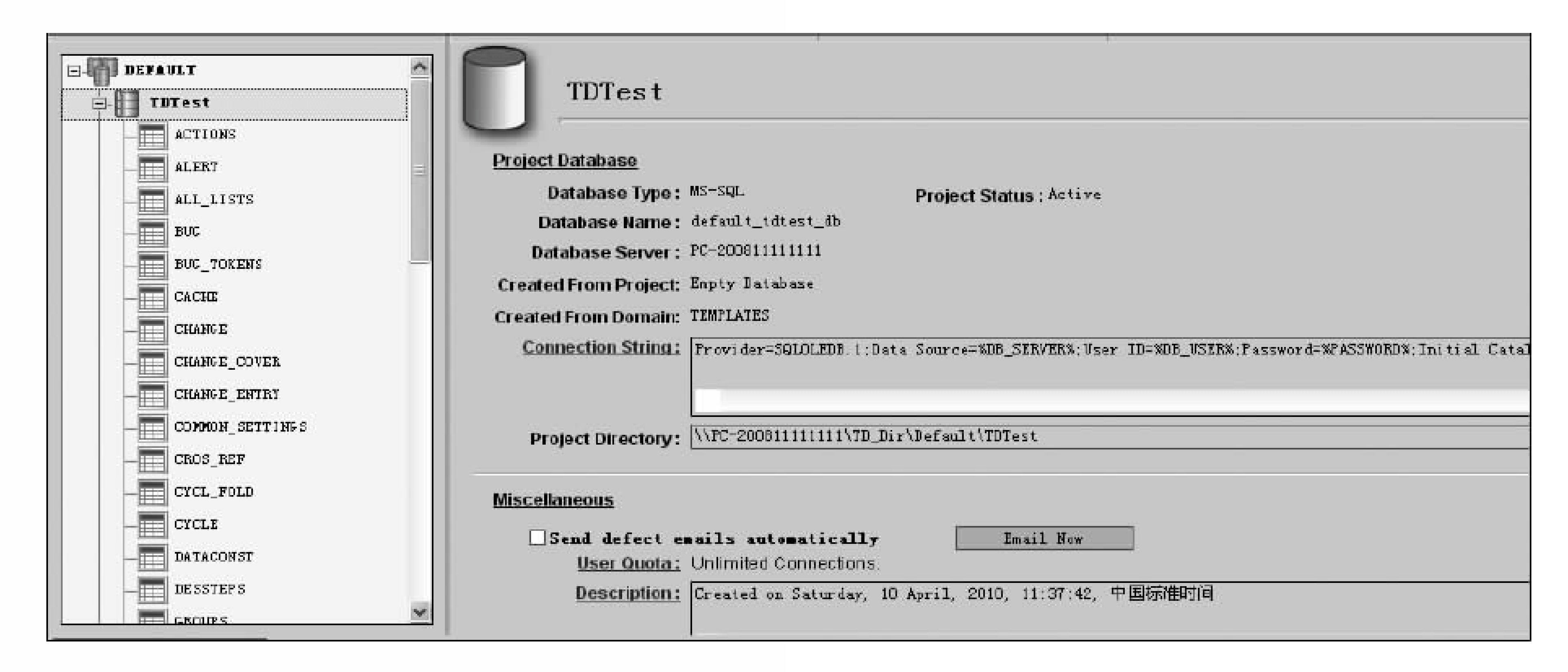


图 11.7　TestDirector 数据库表信息

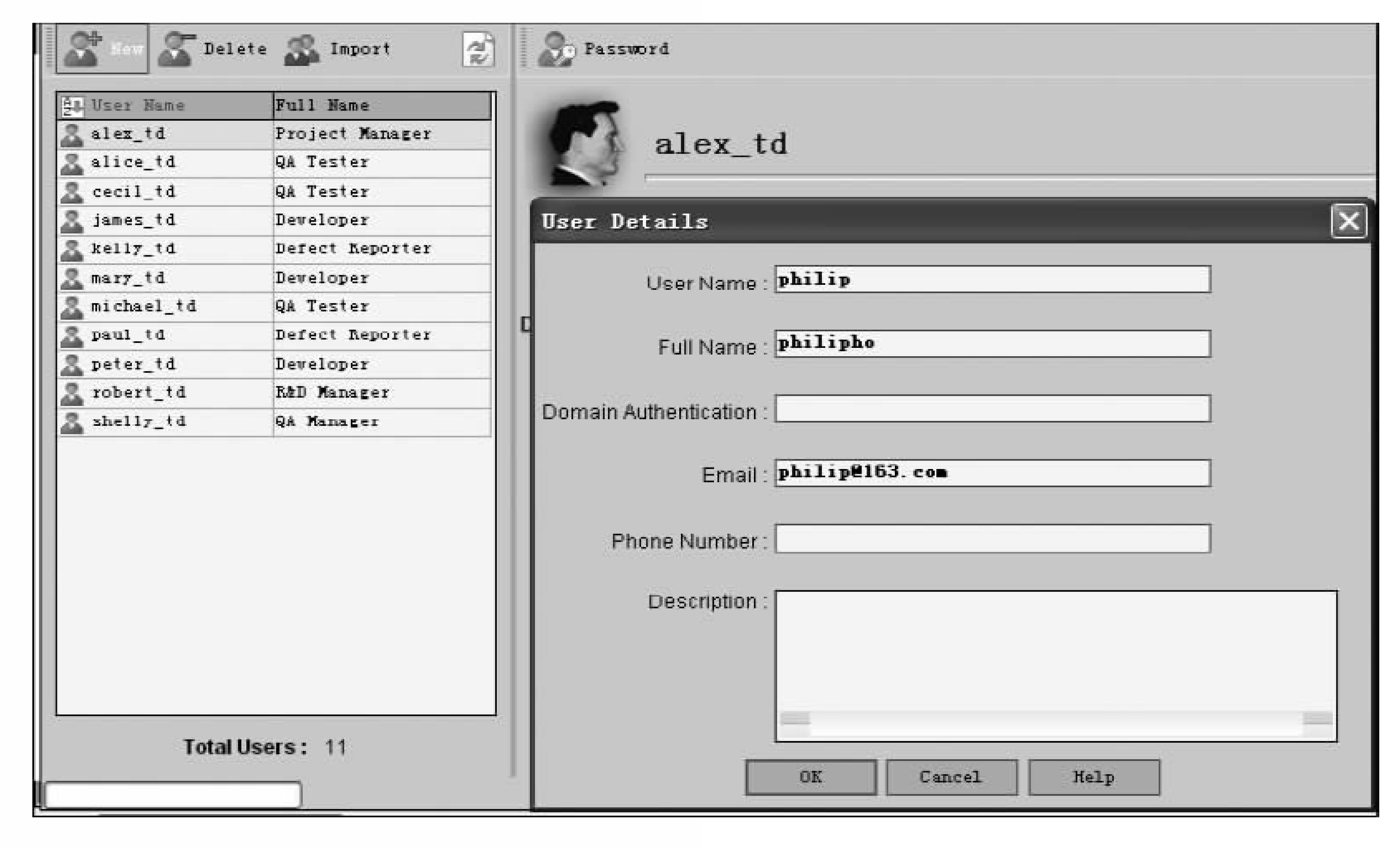


图 11.8　TestDirector 添加用户信息

增加项目用户后，退出 Site Administrator 页面，单击 TD 页面左边连接列表中的 TestDirector 连接进入 TD 登录页面，在 TD 登录页面右上角单击 CUSTOMIZE，如图 11.9 和图 11.10 所示。

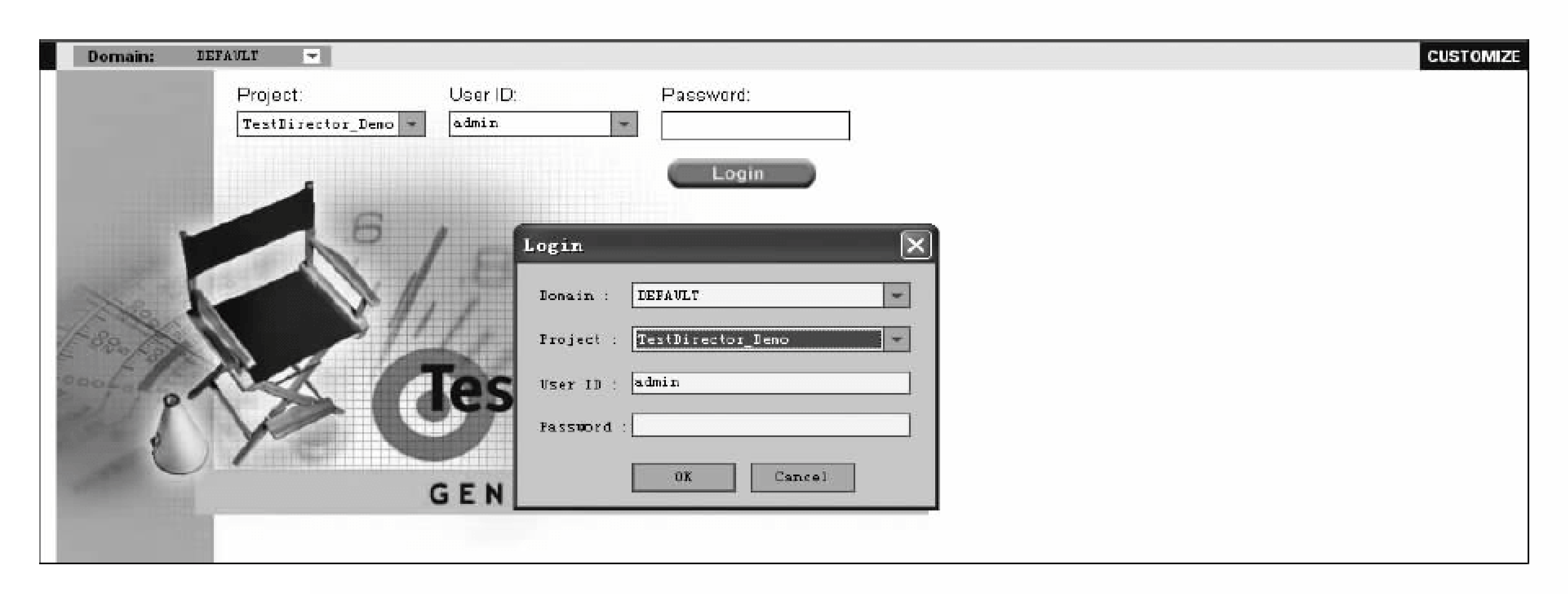


图 11.9　TestDirector 登录 CUSOMIZE

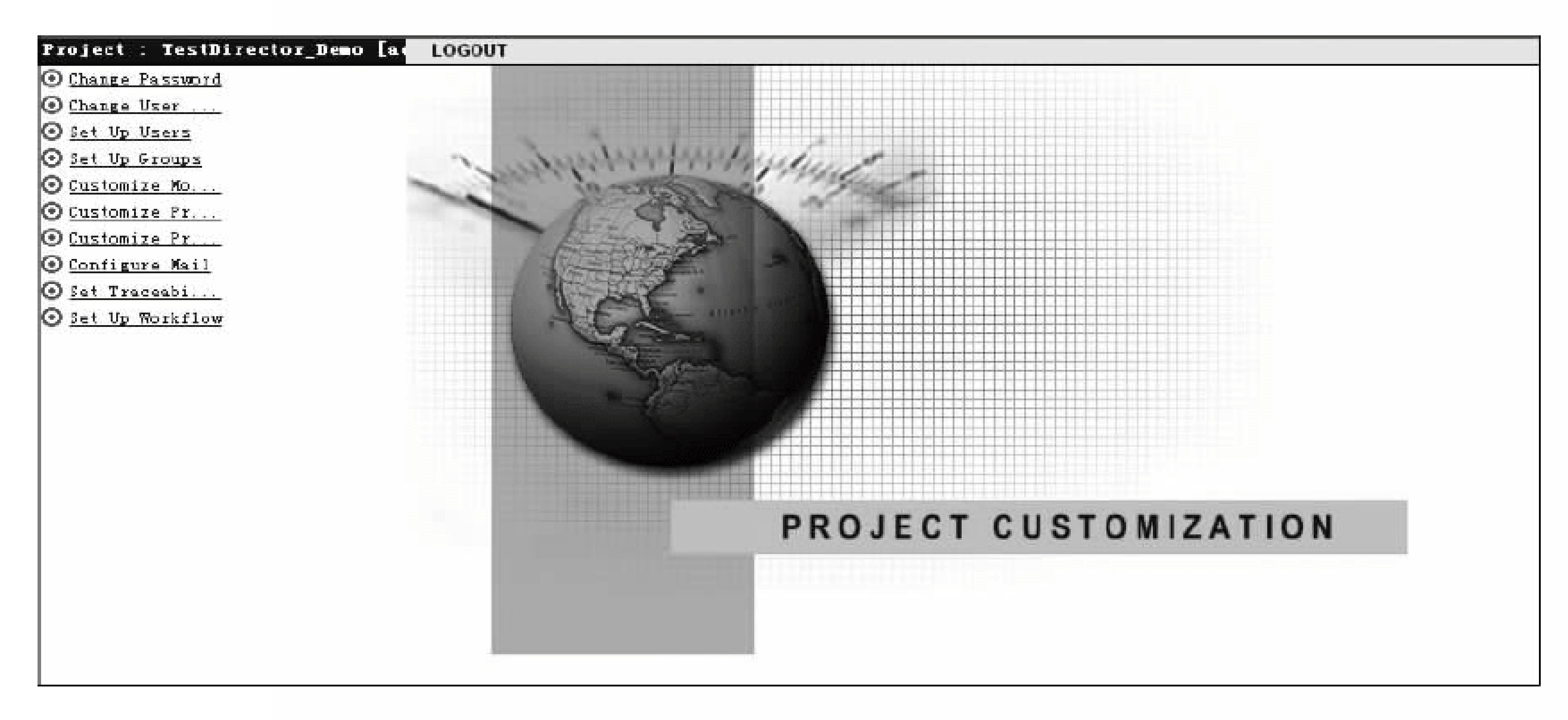


图 11.10　TestDirector 分配权限

TestDirector 提供不同的权限管理：

- TDAdmin：Admin 用户群成员有全部的权限。
- QATester：具有管理需求、测试计划、测试库在缺陷模块等权限，只能添加和修改缺陷，不能删除缺陷。
- Project Manager：具有管理需求、测试计划、测试库和缺陷的权限。
- Developer：具有管理测试库在缺陷模块。
- View：在 TestDirector 项目里只有只读权限，只能更改自己的密码和属性。

2）管理测试需求

在进行测试的执行之前，应该指定测试的需求。测试的需求可以与需求规格说明书上的一一对应，实际上，有些公司就是用 TestDirector 的需求管理模块来管理和维护需求规格的。

测试需求详细描述了应该测试的范围和内容。给测试组的后续测试过程提供了一个

测试的基本依据。测试需求可以与测试的用例和缺陷对应起来，从而实现需求的可跟踪性，并且帮助测试组长和项目经理判断测试的质量，以及提供产品满足需求的程度的参考。应该通过定义测试需求来开始整个应用程序的测试过程。需求详细地描述了应用程序中哪些需要被测试，并为测试组提供了整个测试过程的基础。通过定义这些需求，能够更好地聚焦于商业需要对测试进行计划和管理。需求与测试和缺陷关联，从而确保整个过程可追溯并帮助整个过程的决策。

在 TestDirector 中是通过 REQUIREMENTS 模块的需求树来管理测试需求的，如图 11.11 所示。

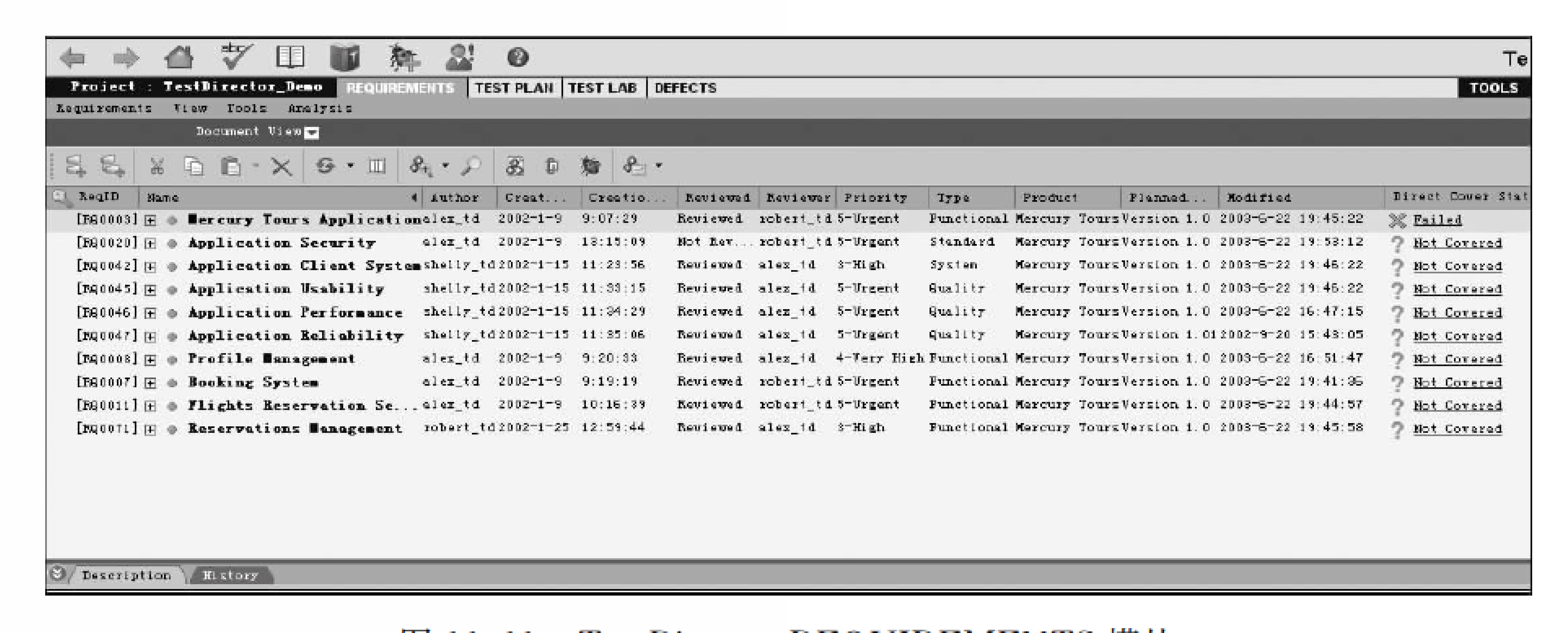


图 11.11　TestDirector REQUIREMENTS 模块

通过选择 Requirements→New Requirement 和 New Child Requirement 来创建需求项和子需求项，组织需求的层次关系，从而形成需求树。

每一项需求都有唯一的“ReqID”标识。在每一个需求项的 Description 页，描述该项需求的具体内容，通过在“Priority”列中为每一个需求项选择一个优先级。默认优先级划分成以下几个级别。

- “1-Low”：低；
- “2-Medium”：中等；
- “3-High”：高；
- “4-Very High”：较高；
- “5-Urgent”：极高。

可根据需要选择合适的优先级别。测试需求与测试用例可以通过测试覆盖把两者关联起来。关联的操作步骤如下：

(1) 切换到 Coverage View 的覆盖视图。

(2) 选择 Document View 所在的下拉框。

(3) 然后选中 Coverage View，则以覆盖视图展示需求项及覆盖的测试用例，如图 11.12 所示。

在需求树中选中某个需求项后，可在右边界面的 Tests Coverage 页面单击 Select Tests 按钮，选择需要的测试用例，当然前提是测试用例已经设计好。这样就把需求和覆

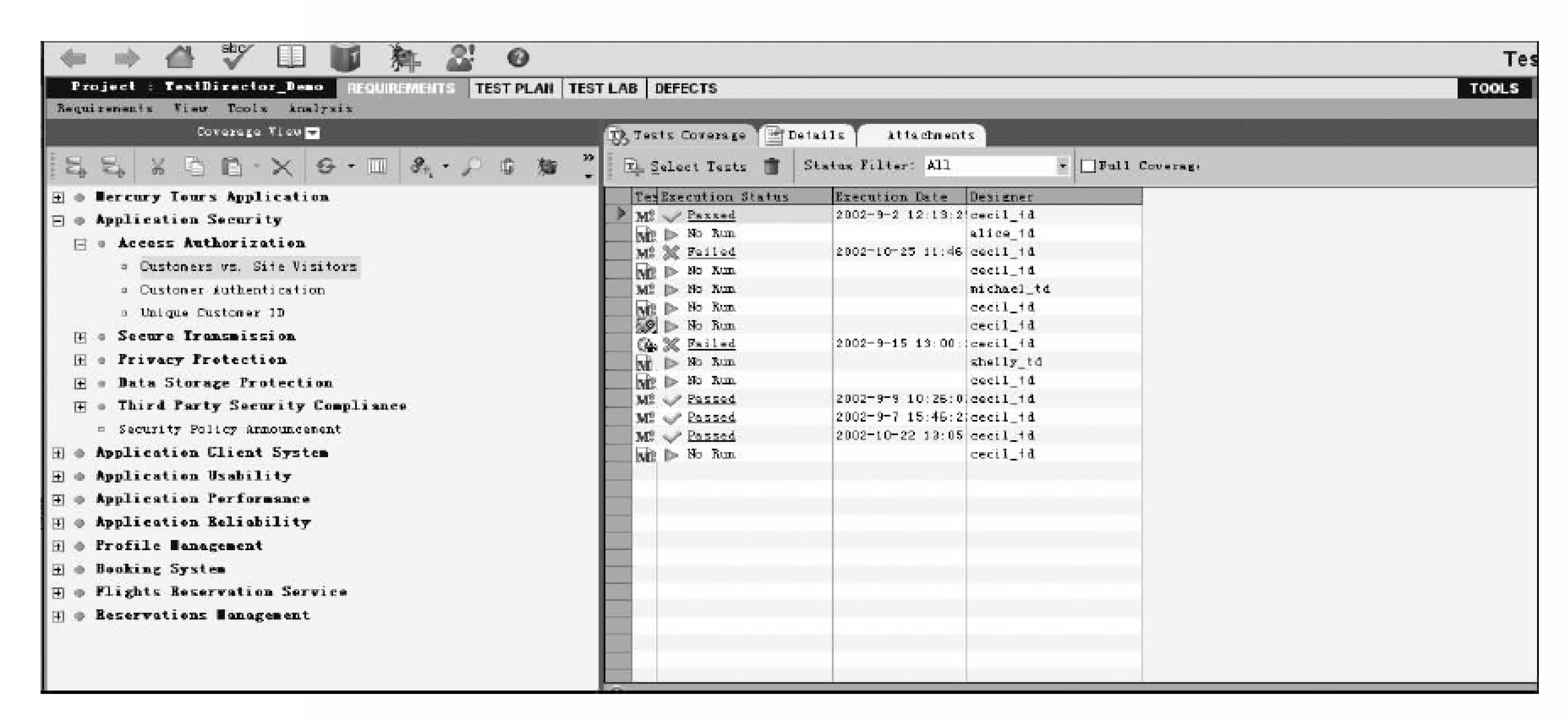


图 11.12 TestDirector Coverage View

盖需求的测试用例关联起来。可以通过计算每一项需求对应的测试用例个数来统计对该项需求的覆盖力度。需求与测试用例之间的关联是相互的,也可以在测试用例设计的界面选择需要覆盖的需求项,来达到关联的目的。这在后面会再讲到。

在测试需求管理模块中,还有生成分析报告的功能,包括测试需求文档报告、需求基本情况报告、需求覆盖情况报告等。图 11.13 展示了需求的优先级分布情况。要生成分析报告,可选择 Analysis 菜单,单击 Reports 子菜单选择需要报告的类型,该项用于产生文本类型的报告;而选择 Graphs 子菜单则用于产生图表类型的报告。

测试人员都知道,最终判断软件是否正确必须依据用户的需求。而 Test Director 就是通过上面所说的测试需求管理模块来管理和维护这一测试的。

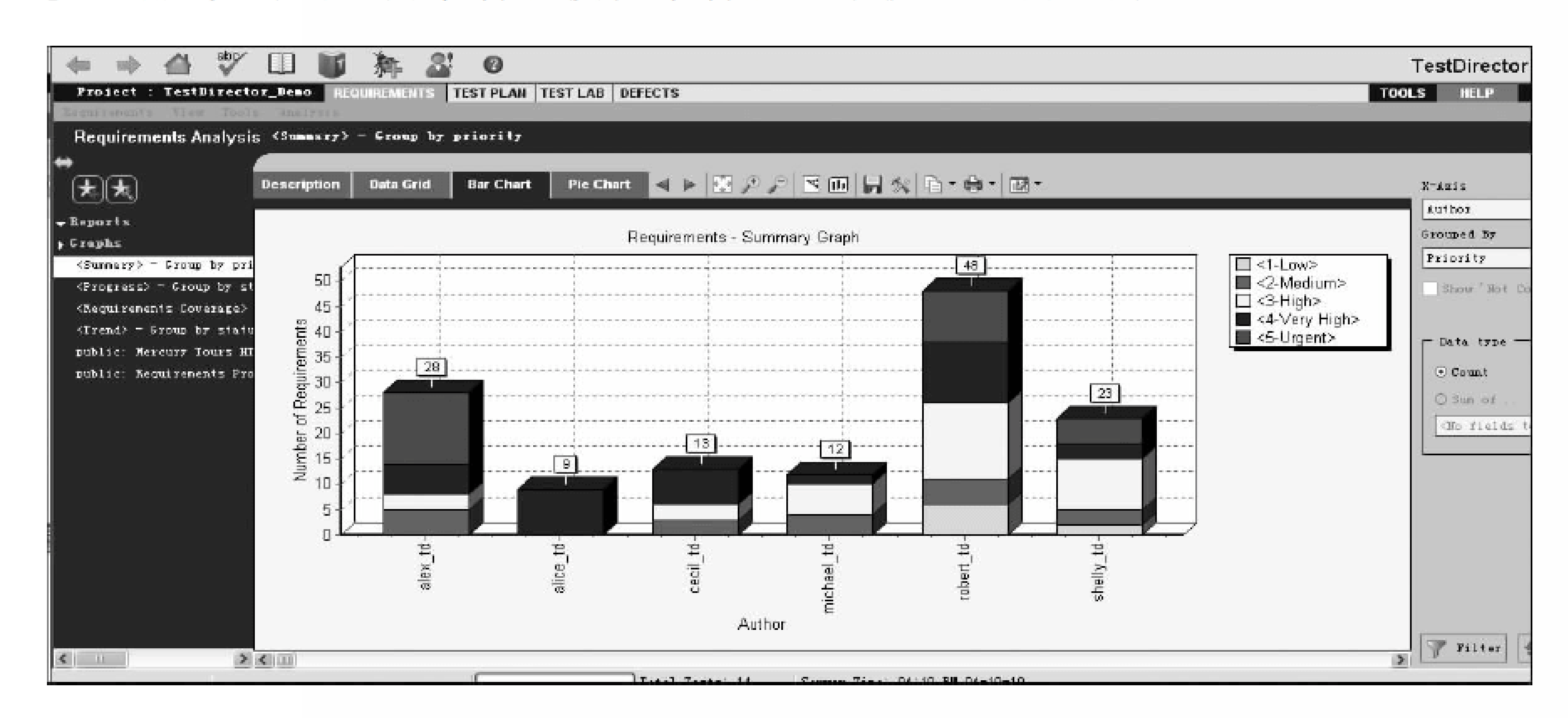


图 11.13 TestDirector 优先级分布图

3) 管理测试用例

在分析和明确了测试需求后,需要进一步地详细设计覆盖需求的测试用例,用于指导测试人员的测试执行。测试用例的设计是对将来的测试活动进行计划的一个过程, Test

Director 通过 TEST PLAN 模块来管理测试用例。测试用例的管理可以通过树状结构的层次关系来组织，如图 11.14 所示。

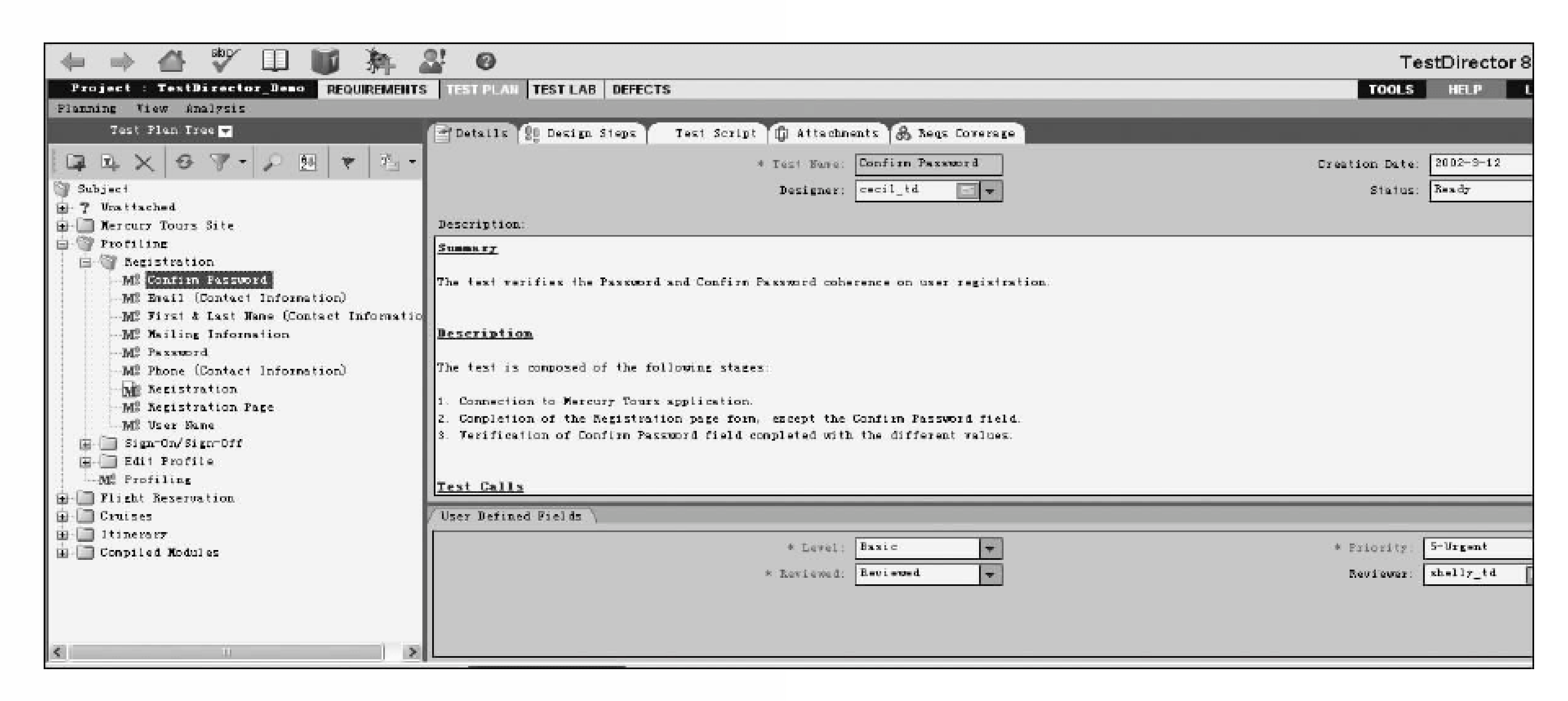


图 11.14　TestDirector 测试用例管理界面

下面是一个添加测试用例的操作过程。

(1) 选择 Planning→New Folder→New Test 命令，来创建测试用例分类目录和测试用例。

(2) 选中某个测试用例，可在右边的 Details 页中编辑对该测试用例的描述，例如测试用例的目的、测试用例输入的参数等。

(3) 在 Design Steps 页中可编辑测试用例的测试步骤、预期结果，如图 11.15 所示。

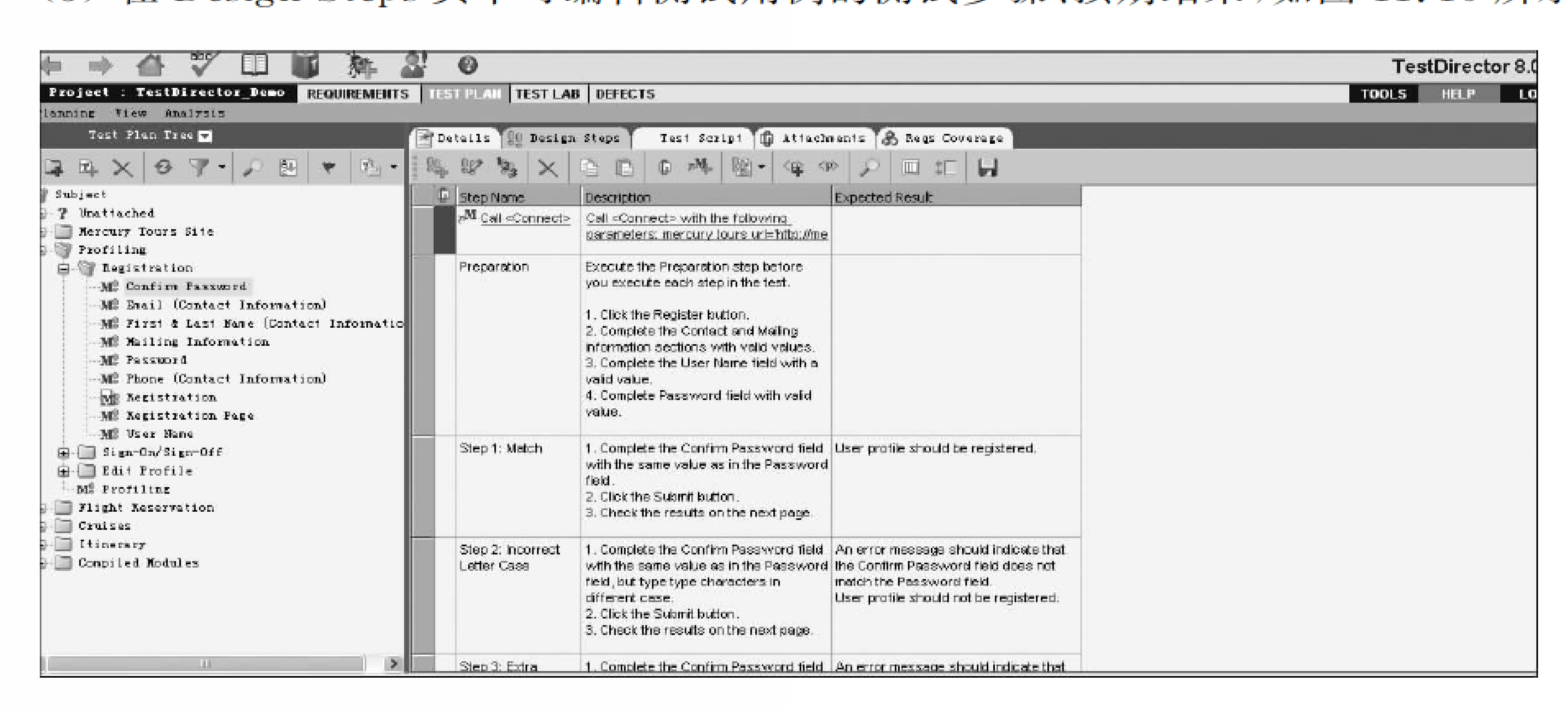


图 11.15　TestDirector 测试用例中的测试步骤

(4) 在 Attachments 页中可以加入一些必要的附件，例如，测试参数文件或测试数据文件，还可以加入 URL 地址、屏幕截图、系统信息等内容。

(5) 在 Reqs Coverage 页中可以单击 Select Req，加入本测试用例覆盖的需求项，如图 11.16 所示。

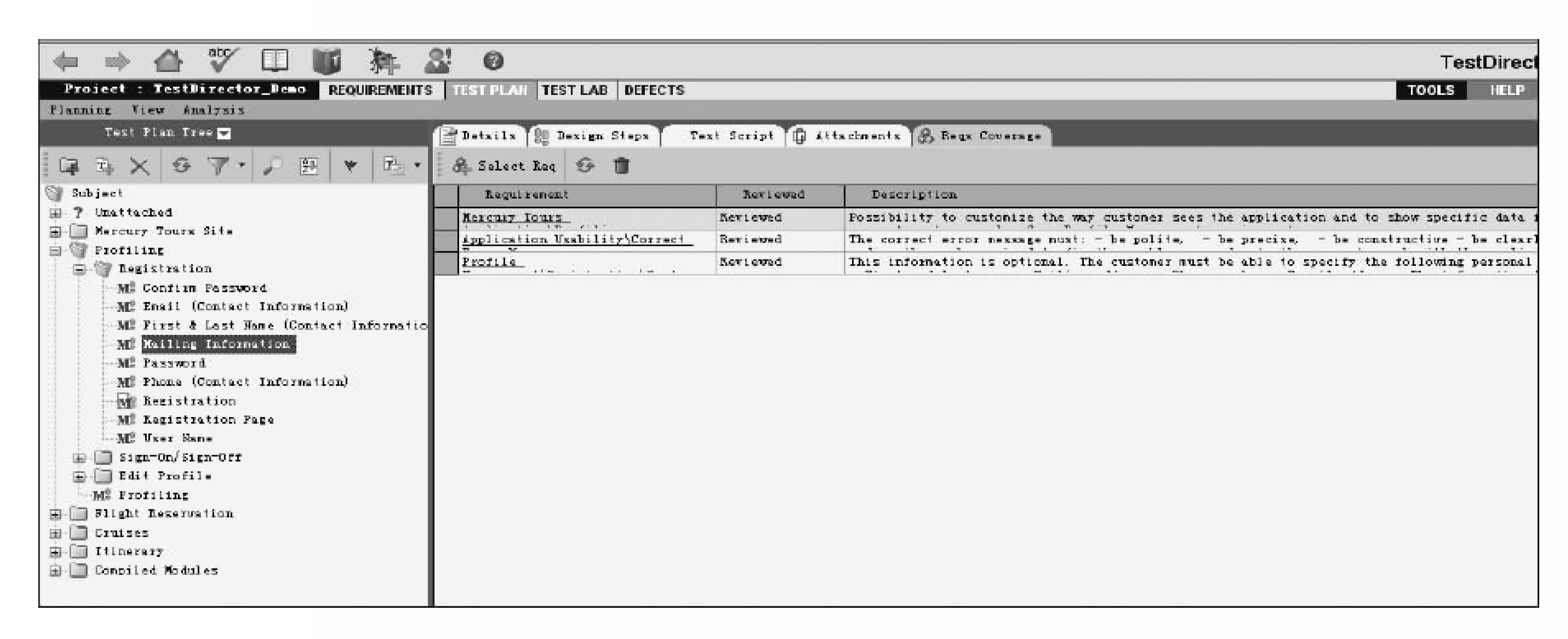


图 11.16　TestDirector 测试用例中的覆盖项

在测试用例设计模块中，可通过选择 Analysis 命令，来选择各种关于测试用例的报告。测试用例是测试的设计结果，TestDirector 通过测试计划模块的功能来管理和维护测试用例，形成测试项目的测试用例库。

4）管理测试过程

在设计好测试用例后，就可以在测试执行时选择需要参与测试的测试用例，打包成一个测试的集合，并按照集合中的每一项测试用例进行测试。这个过程的管理在 Test Director 中是用 TEST LAB 模块来实现的，如图 11.17 所示。

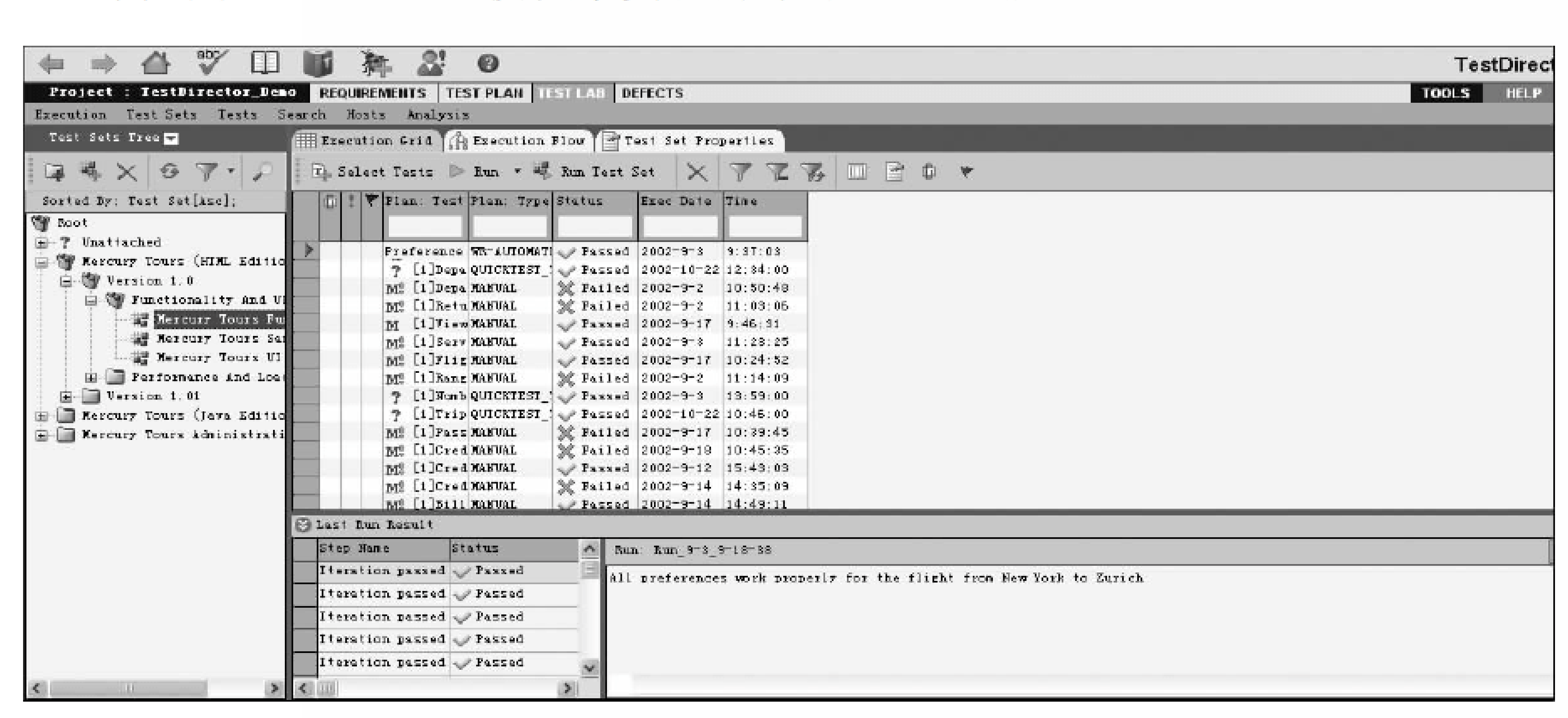


图 11.17　TestDirector 测试管理界面

下面是一个创建测试过程并执行测试的操作步骤：

（1）在界面的左边是 Test Sets 编辑区域，用于添加需要执行的测试集合。单击 Test Sets→New Test Set 命令，就可以添加一个空的测试集合。这是一个计划测试的任务的过程，一般有测试组长负责编辑测试集合，并选择需要在这次测试任务中执行的测试用例。

（2）在右边区域的 Execution Grid 页，单击 Select Tests 按钮，可为测试集合添加需

要被执行的测试用例。

(3) 添加了需要的测试用例后，测试组长在 Responsible Tester 列中指定测试用例由谁执行。

(4) 通过在 Planned Exec Date 列中指定计划执行的时间，如图 11.18 所示。

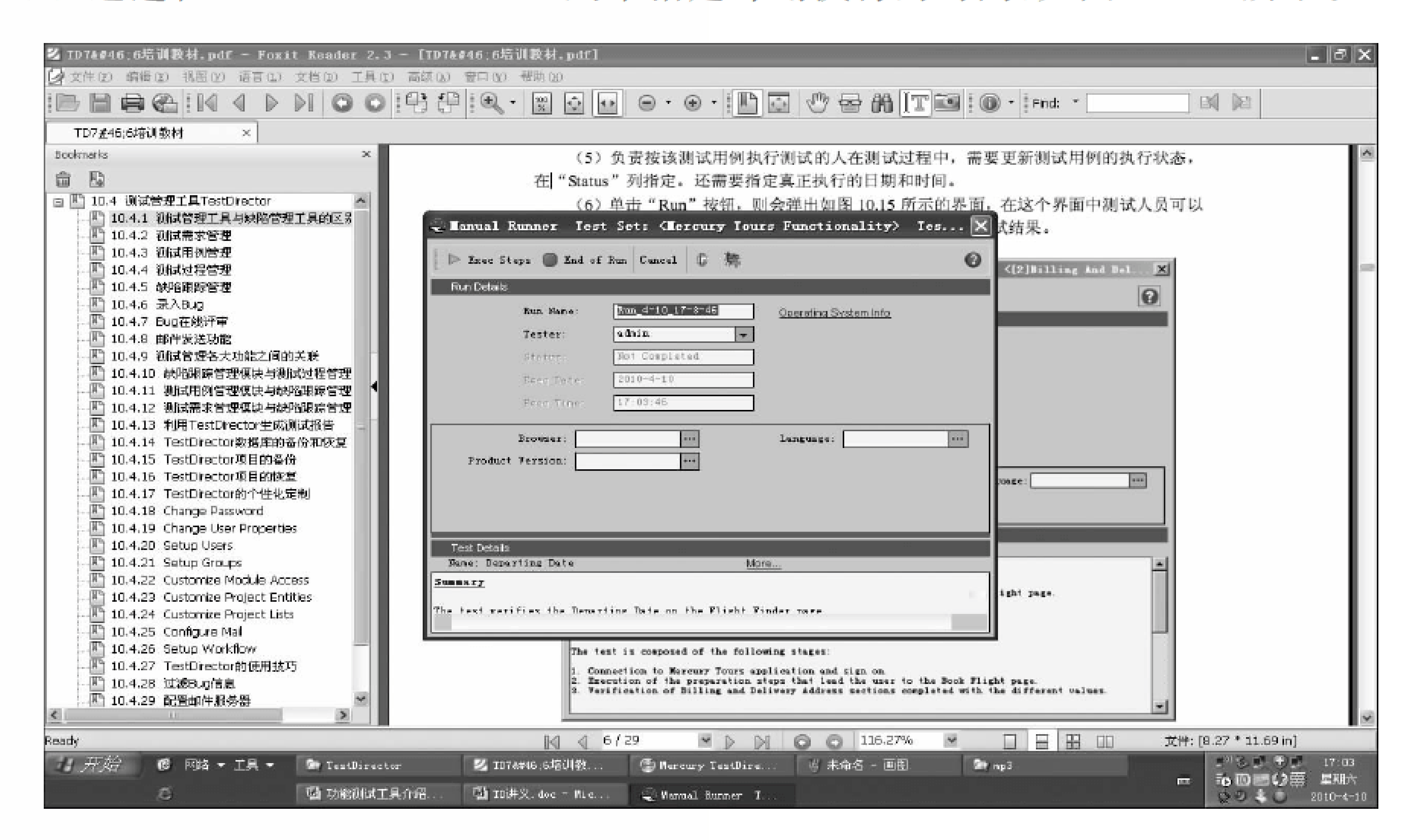


图 11.18　TestDirector 测试执行界面

(5) 负责按该测试用例执行测试的人在测试过程中，需要更新测试用例的执行状态，在 Status 列指定。还需要指定真正执行的日期和时间。

(6) 单击 Run 按钮，一步一步地执行测试，并登记测试结果。

在测试过程管理模块，可通过选择 Analysis 菜单，来选择各种关于测试过程和进度的报告。测试过程管理是考验测试人员，尤其是测试管理者的安排测试任务和分工的技巧、选择测试用例的策略。测试过程也是测试工作的记录过程，是测试人员工作过程的体现。

5) 管理测试缺陷

缺陷跟踪管理功能是测试管理工具的主角，也是测试人员日常工作最常待的地方，并且也是测试人员与开发人员沟通的基础平台。缺陷跟踪管理功能在 Test Director 中是通过 DEFECTS 模块来实现的，如图 11.19 所示。

6) 录入 Bug

单击 Defects→Add Defect 命令，可以往缺陷列表中添加一条 Bug 记录，如图 11.20 所示，在这个界面上，测试人员可以将发现的 Bug 的相关信息录入。

一般需要录入的信息应该包括以下字段的内容。

- Summary：Bug 的简要描述。
- Assigned To：分派给谁处理这个 Bug。
- Detected By：这个 Bug 是谁发现的。

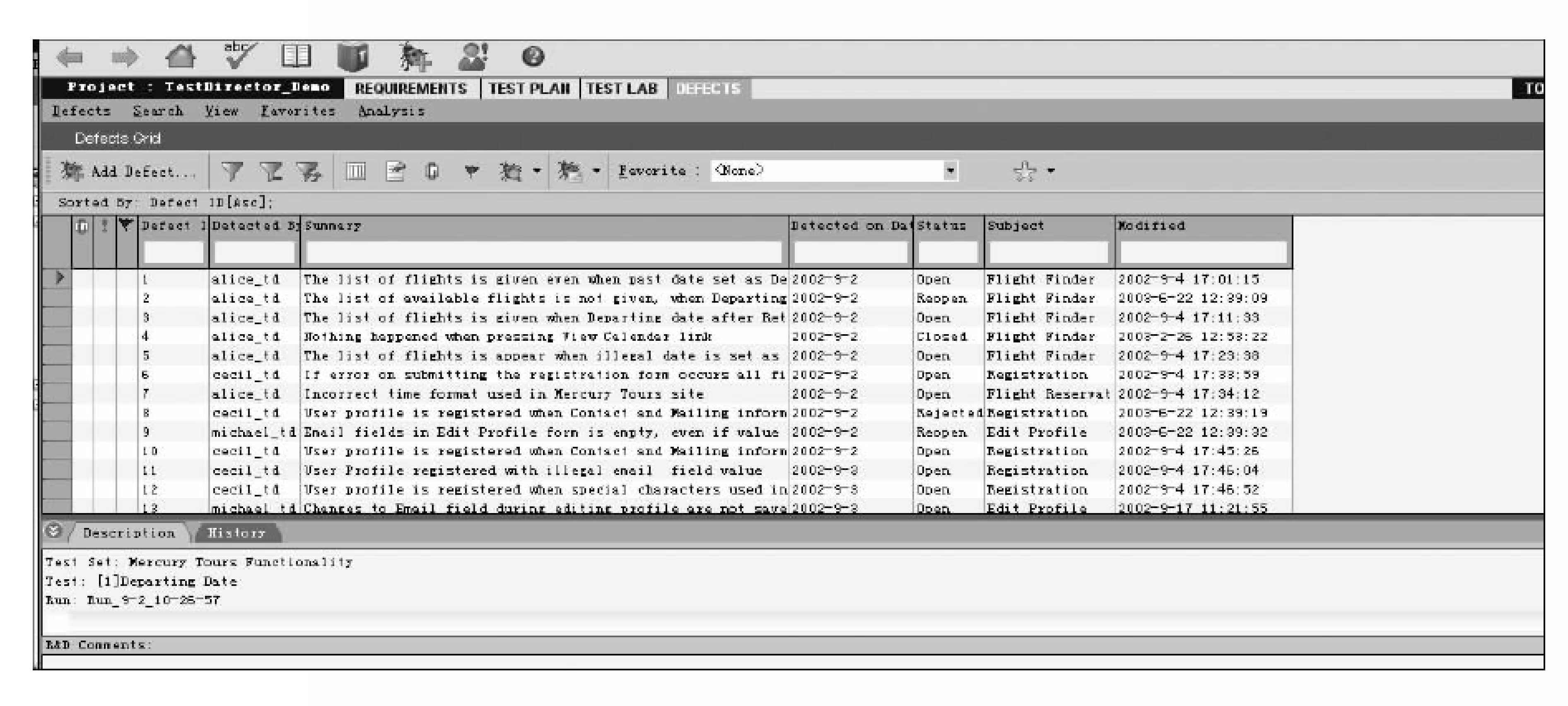


图 11.19　TestDirector 缺陷跟踪管理界面

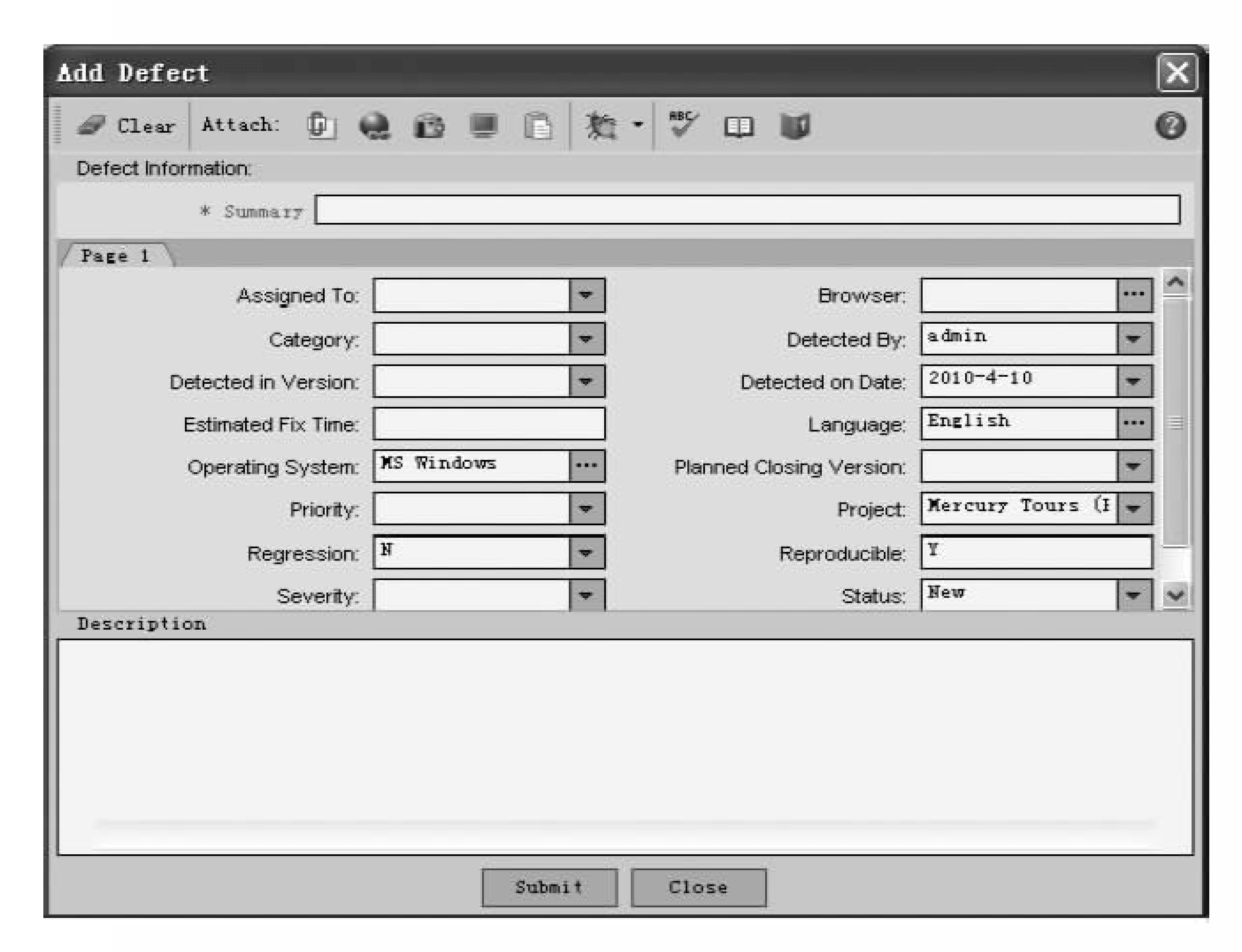


图 11.20　Bug 录入界面

- Detected in Version：这个 Bug 是在哪个版本发现的。
- Severity：Bug 的严重级别。
- Project：Bug 出现在哪个项目。
- Subject：Bug 出现在哪个功能模块。
- Status：一般新录入 Bug 时把 Bug 状态设置为 New。
- Description：Bug 的详细描述。

把 Bug 正确录入后，接下来的工作就是跟踪 Bug 的修改状态直到关闭。不同的项目组成员或角色可以使用不同的功能来对 Bug 记录进行管理和维护。

7）利用 TestDirector 生成测试报告

在 TestDirector 的每个模块中，都会有一个 Analysis 的菜单，用于查看和定制各种

类型的分析报告。这些功能都分成两大类报告，一类是从 Reports 子菜单进去的报告，另一类是从 Graphs 子菜单进去的报告。这两类报告代表了“记录型”和“分析型”两种测试报告。

（1）记录型的报告是指把所有过程记录数据罗列出来，形成列表。在测试需求管理模块中，选择 Reports→Standard Requirements Report，则会出现如图 11.21 所示的报告。

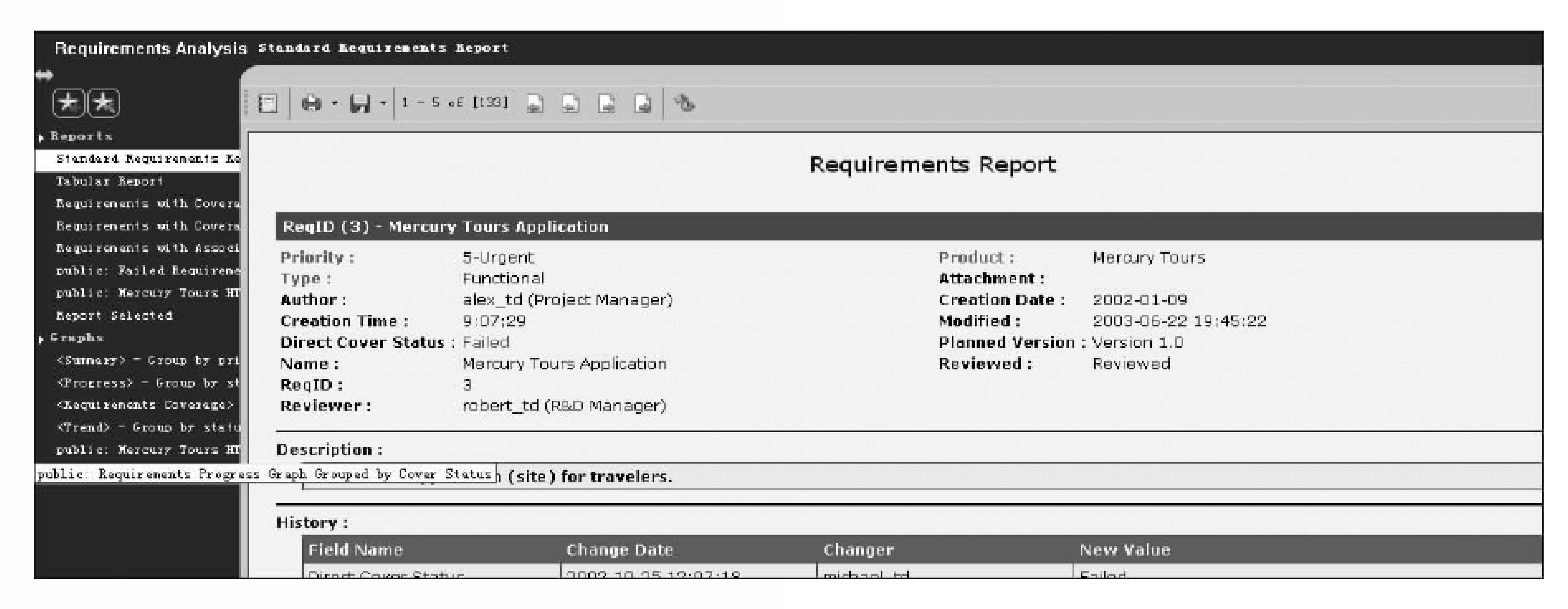


图 11.21　标准需求报告

（2）分析型的报告是指通过统计图表、趋势图、状态图等形式对数据进行了分析和统计。这种报告的特点是能让别人看到整体的、全局的、高层的、形象的、直观的报告。例如，在测试用例管理模块，选择 Graphs→＜Summary-Group by Status＞，则出现如图 11.22 所示的报告。

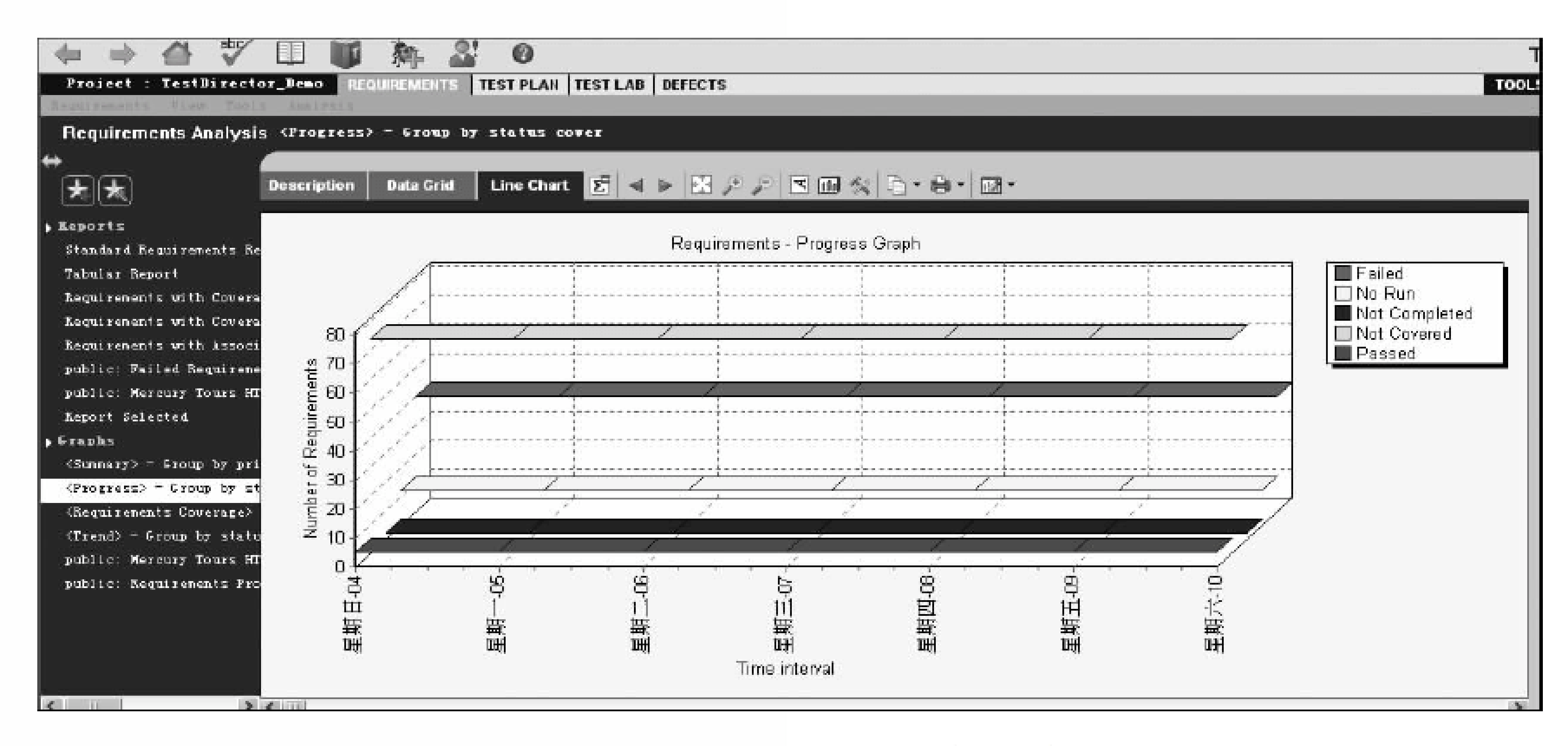


图 11.22　测试人员的测试用例设计情况

两种类型的报告各有特点，全面的测试报告应该包括两大类型的报告。让不同的项目组成员可以找到自己关心的报告区域。例如，项目经理可能更关心高层的分析总结性的数据，而开发人员和测试人员可能需要仔细分析基础性的数据。



性能测试工具

性能测试工具有 Radview 公司的 WebLoad、Microsoft 公司的 WebStress 等;针对数据库测试的 TestBytes;对应用性能进行优化的 EcoScope 等工具。MercuryInteractive 的 LoadRunner 是一种适用于各种体系架构的自动负载测试工具,其测试对象是整个企业的系统,通过模拟用户的操作行为进行实时监测。

12.1 综　　述

Mercury LoadRunner 是一种预测系统行为和性能的负载测试工具。通过以模拟上千万用户实施并发负载及实时性能监测的方式来确认和查找问题,LoadRunner 能够对整个企业架构进行测试。通过使用 LoadRunner ,企业能最大限度地缩短测试时间,优化性能和加速应用系统的发布周期。

下面简单介绍 LoadRunner 的相关术语。

1. 虚拟用户生成器(VuGen)

虚拟用户生成器用于捕获最终用户业务流程和创建自动性能测试脚本。VuGen 通过录制应用程序中典型最终用户执行的操作来生成虚拟用户。VuGen 将这些操作录制到自动虚拟用户脚本中,以便作为负载测试的基础。

2. Controller

Controller 用于组织、驱动、管理和监控负载测试。Controller 是用来创建、管理和监控负载测试的中央控制台。使用 Controller 可以运行用来模拟真实用户执行的操作的脚本,并可以通过让多个虚拟用户同时执行这些操作来在系统中创建负载。

3. 负载生成器

负载生成器用于通过运行虚拟用户生成负载。

4. Analysis

Analysis 用于查看、分析和比较性能结果。Mercury Analysis 提供包含深入的性能分析信息的图和报告。使用这些图和报告,可以标识和确定应用程序中的瓶颈,并确定需要对系统进行哪些更改来提高系统性能。

5. 场景

场景用于根据性能要求定义在每一个测试会话运行期间发生的事件。

6. Vuser

在场景中，LoadRunner 用虚拟用户代替实际用户。Vuser 模拟实际用户的操作来使用应用程序。一个场景可以包含几十、几百甚至几千个 Vuser。

12.2 LoadRunner 测试流程

负载测试通常由五个步骤组成：计划、脚本创建、场景定义、场景执行和结果分析。

1. 计划

定义性能测试要求，例如并发用户的数量、典型业务流程和所需响应时间。在任何类型的测试中，测试计划都是必要的步骤。测试计划是进行成功的负载测试的关键。任何类型的测试的第一步都是制订比较详细的测试计划。一个比较好的测试计划能够保证 LoadRunner 完成负载测试的目标。

2. 创建脚本

LoadRunner 使用虚拟用户的活动来模拟真实用户来操作 Web 应用程序，而虚拟用户的活动就包含在测试脚本中，开发测试脚本要使用 VuGen 组件。测试脚本要完成的内容有：

- 每一个虚拟用户的活动。
- 定义结合点。
- 定义事务。

3. 定义场景

使用 LoadRunner Controller 设置负载测试环境。

4. 运行场景

通过 LoadRunner Controller 驱动、管理和监控负载测试。

5. 分析结果

使用 LoadRunner Analysis 创建图和报告并评估性能。

12.3 项目实践

LoadRunner8.1 安装结束后，自带 Flight Reservation（预订航班）系统作为测试示例。预订航班系统内容如下：一个基于 Web 的旅行代理应用程序，并要确定多个用户同

时执行相同的事务时,该应用程序将如何处理。使用 LoadRunner 代替旅行代理,您可以创建具有 1000 个 Vuser 的场景,并且这些 Vuser 可以同时尝试 在应用程序中预订航班。

LoadRunner 测试过程由以下四个基本步骤组成:

① 创建脚本:捕获在您的应用程序中执行的典型最终用户业务流程。

② 设计场景:通过定义测试会话期间发生的事件,设置负载测试环境。

③ 运行场景:运行、管理并监控负载测试。

④ 分析结果:分析负载测试期间 LoadRunner 生成的性能数据。

12.3.1 使用 VuGen 创建脚本

创建负载测试的第一步是使用 VuGen 录制典型最终用户的业务流程。VuGen 采用录制/回放机制。当用户在应用程序中按照业务流程操作时,VuGen 将这些操作录制到自动脚本中,以便作为负载测试的基础。

在此部分中,将录制旅行代理为一位乘客预订从丹佛到洛杉矶的航班的流程。

1. 准备录制

开始先打开 VuGen 并创建一个空白脚本。

1) 启动 LoadRunner

选择 “开始”→“程序”→ Mercury LoadRunner → LoadRunner,打开 Mercury LoadRunner 8.1 开始界面,如图 12.1 所示。

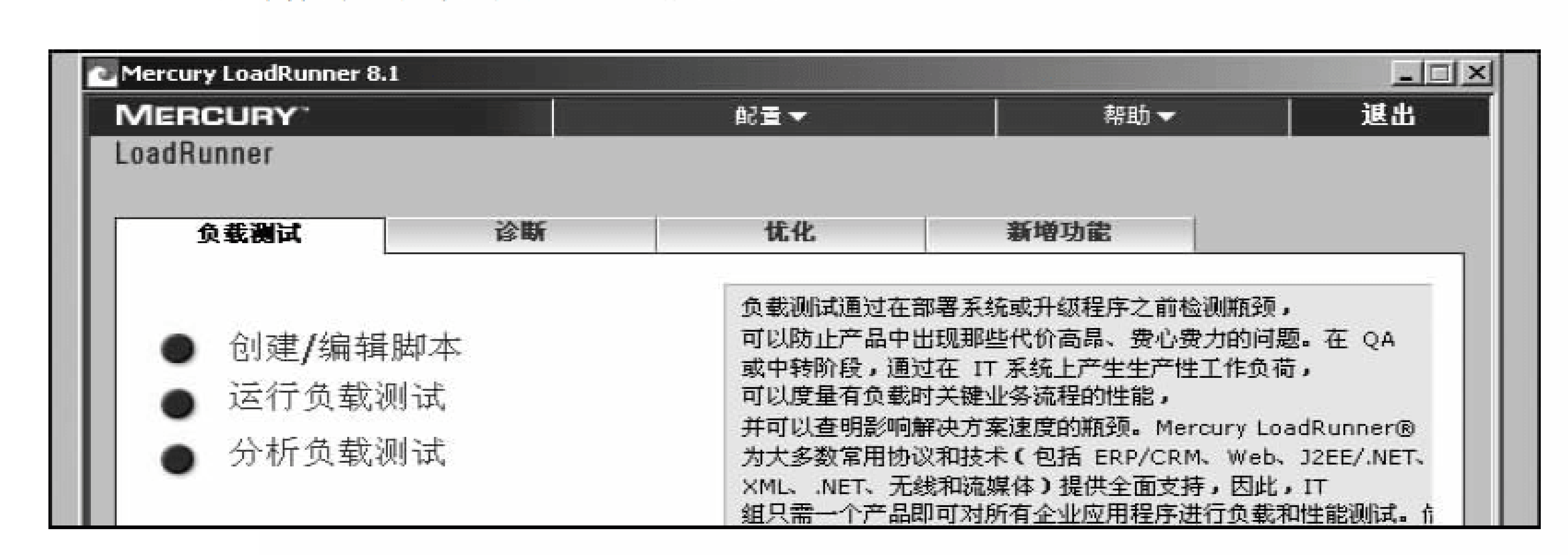


图 12.1　LoadRunner 开始界面

2) 打开 VuGen

在“负载测试”选项卡中单击“创建/编辑脚本”,打开 VuGen 的开始界面。如图 12.2 所示。

3) 创建一个空白 Web 脚本

在 VuGen 的开始界面中的“脚本”选项卡中,单击“新建 Vuser 脚本”,将打开“新建虚拟用户”对话框,并显示“新建单协议脚本”屏幕,如图 12.3 所示。

确保“类别”类型为“所有协议”。VuGen 将显示所有可用于单协议脚本的协议列表。向下滚动查看该列表,选择“Web(HTTP/HTML)”,并单击“确定”按钮,创建一个空白 Web 脚本。

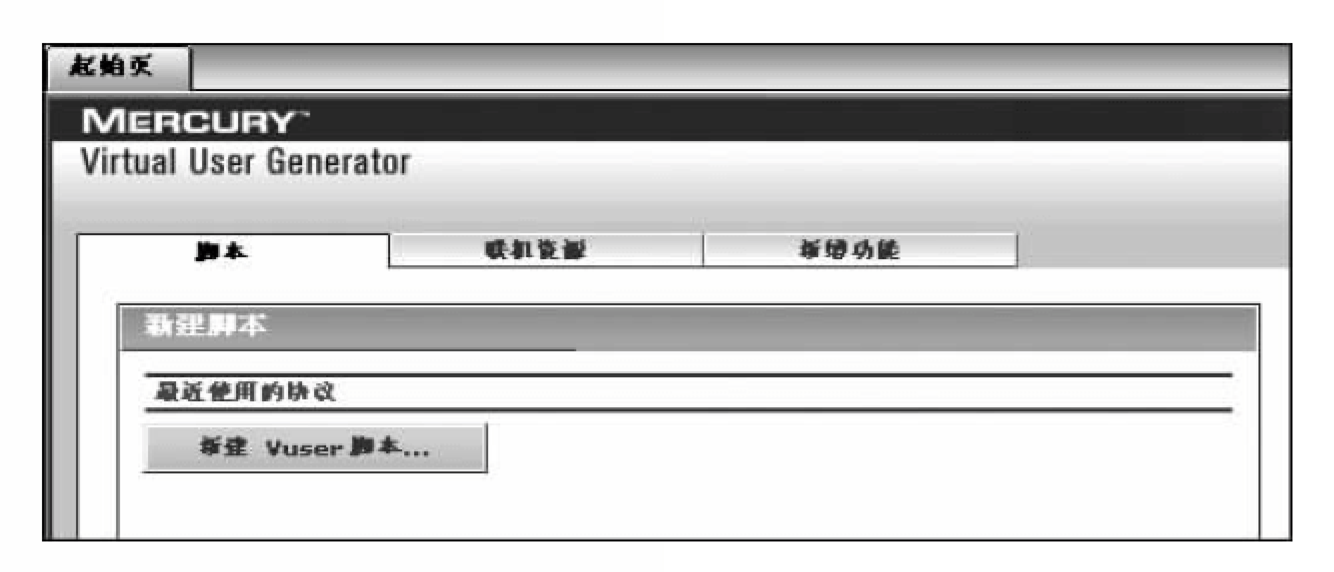


图 12.2　VuGen 界面



图 12.3　LoadRunner 新建虚拟用户界面

2. 使用 VuGen 向导录制业务流程

空脚本以 VuGen 的向导模式打开，且任务窗格显示于左侧（如果未显示任务窗格，请单击工具栏上的“任务”按钮）。VuGen 的向导将指导用户逐步完成创建脚本，然后根据用户的测试环境进行相应修改的过程。任务窗格列出了脚本创建过程中的每个步骤或任务。在逐步完成每一步操作的过程中，VuGen 会在窗口的主区域显示详细的说明和准则，如图 12.4 所示。

具体录制脚本过程如下：

（1）在 Mercury Tours 网站上开始录制。

在任务窗格中，单击步骤①中的“录制应用程序”。单击说明窗格底部的“开始录制”，如图 12.5 所示。

在 URL 地址框中，输入 http://localhost:1080/MercuryWebTours/。在“录制到操作”下拉文本框中，选择“操作”。单击“确定”按钮。将打开一个新的 Web 浏览器，并显示 Mercury Tours 站点。如果在打开站点时出现错误，请确保 Web 服务器在运行。要启动

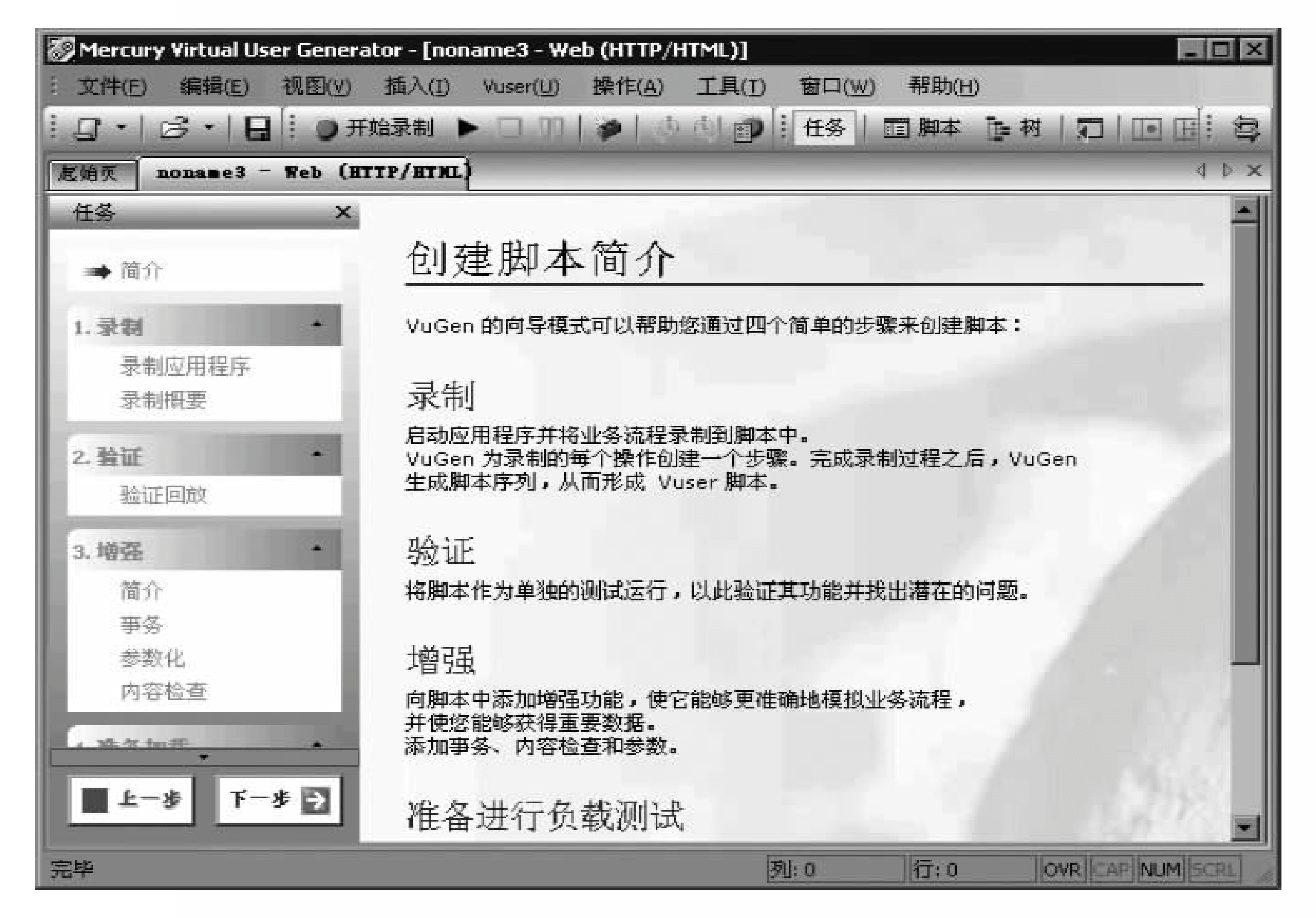


图 12.4 VuGen 界面

服务器，请选择“开始”→“程序”→Mercury LoadRunner→“示例”→Web→“启动 Web 服务器”。将打开浮动的“录制”工具栏，如图 12.6 所示。



图 12.5 VuGen 录制对话框



图 12.6 录制工具栏

(2) 登录到 Mercury Tours 网站。

输入账号和密码，单击“登录”。将打开欢迎页面。

(3) 输入航班详细信息。

单击“航班”。将打开“查找航班”页，在该页面上填写以下内容：

- 出发城市：丹佛（默认设置）；
- 出发日期：保持默认设置不变（当前日期）；
- 到达城市：洛杉矶；
- 返回日期：保持默认设置不变（第二天的日期）。

保持其余的默认设置不变，然后单击“继续”按钮，打开“搜索结果”页。

(4) 选择航班。

单击“继续”按钮接受默认航班选择，将打开“付费详细信息”页。

(5) 输入付费信息并预订航班。

单击“继续”按钮。打开 “发票”页,并显示发票。

(6) 查看路线。

在左窗格中单击“路线”按钮。将打开 “路线”页。

(7) 在左窗格中单击 “注销”。

(8) 单击浮动工具栏上的 “停止”以停止录制过程。

一旦生成了 Vuser 脚本,Vuser 向导将自动前进到任务窗格中的下一步,并显示包含协议信息以及在会话期间创建的一系列操作的录制概要。对于录制期间执行的每个步骤,VuGen 都生成一个快照(即录制期间各窗口的图片)。这些录制的快照的缩略图显示在右侧窗格中。

(9) 选择“文件”→“保存”,或单击“保存”。在“文件名”框中输入 basic_tutorial。单击“保存”。VuGen 将把该文件保存在 LoadRunner 脚本文件夹中,并在标题栏中显示该测试名称。

3. 查看脚本

现在,可在树视图或脚本视图中查看脚本。树视图是基于图标的视图,其中将 Vuser 的操作作为步骤列出;而脚本视图是基于文本的视图,其中将 Vuser 的操作作为函数列出。

1) 树视图

树视图中查看脚本可选择“查看”→“树视图”或单击“树视图”按钮。对于录制期间执行的每个步骤,VuGen 都在测试树中生成了一个图标和一个标题,如图 12.7 所示。



图 12.7 树视图

在树视图中,其中将用户的操作作为脚本步骤列出。大多数步骤都附带相应的录制快照。

2) 脚本视图

脚本视图是基于文本的视图,其中将 Vuser 的操作作为 API 函数列出。选择“查看”→“脚本视图”或单击“脚本视图”按钮,如图 12.8 所示。

在脚本视图中,VuGen 在编辑器中通过彩色编码函数及其参数值显示脚本。用户可以直接在此窗口输入 C 或 LoadRunner API 函数以及控制流语句。

```
Action()
{
    web_url("newtours.mercury.com",
        "URL=http://newtours.mercury.com/",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t1.inf",
        "Mode=HTML",
        LAST);
    lr_think_time(2);
    web_link("new location",
        "Text=new location",
        "Snapshot=t2.inf",
        LAST);
```

图 12.8　脚本视图

4. 回放脚本

完成录制后，就可以回放脚本，以便验证它是否准确地模拟了您录制的操作。

(1) 确保显示了任务窗格(如果没有，请单击工具栏中的“任务”按钮)。单击任务窗格中的“验证回放”，然后单击说明窗格底部的“开始回放”按钮。

(2) 如果打开了“选择结果目录”对话框，询问要存储结果目录的位置，请接受默认名称并单击“确定”按钮。

(3) 单击任务窗格中的“验证回放”查看回放概要。回放概要列出了可能检测到的所有错误并显示录制和回放快照的缩略图，可以通过“运行时设置”模拟各种不同类型的用户行为。例如，您可以模拟一个对服务器立即做出响应的用户，也可以模拟一个在做出响应之前先停下来思考的用户

5. 增强脚本

准备负载测试过程时，LoadRunner 允许用户增强脚本以使其更好地反映真实情况。例如，用户可以在脚本中插入名为内容检查的步骤，以验证某些特定内容是否显示在返回页上。用户可以修改脚本来模拟多用户行为，也可以指示 VuGen 度量特定的业务流程。

准备要部署的应用程序时，您需要度量特定业务流程的持续时间，如登录、预订航班等花费的时间。这些业务流程通常由脚本中的一个或多个步骤或操作构成。在 LoadRunner 中，可以通过将想要度量的操作标记为事务来指定一系列操作。本部分将在脚本中插入一个事务以度量用户查找和确认航班所花费的时间。

1) 打开事务创建向导

确保显示了任务窗格(如果没有，请单击“任务”按钮)。在任务窗格的“增强功能”标题下，单击“事务”。将打开事务创建向导。事务创建向导显示脚本中不同步骤的缩略图。

单击“新建事务”按钮。现在，可以拖动事务标记并将其放置在脚本中的指定点。向导提示您插入事务的起始点，如图 12.9 所示。

2) 插入开始事务标记和结束事务标记

使用鼠标将标记放置到标题为搜索航班按钮的第三个缩略图之前并单击。向导会提示用户插入结束点。



名称	TPS	通过	失败	停止
book_flight	0.0	4	0	0
search_flights	0.0	4	0	0
vuser_end_Transaction	0.0	2	0	0
vuser_init_Transaction	0.0	2	0	0
BookFlight_Transaction_Transaction	0.0	4	0	0
logon	0.0	4	0	0

图 12.9　新建事务

3）指定事务的名称

向导提示输入事务的名称。输入 find_confirm_flight，通过将标记拖动到脚本中的其他点来调整事务的起始点或结束点。

12.3.2　使用 Controller 设计场景

使用 Controller 可以将应用程序性能测试需求划分为多个场景。场景定义每个测试会话中发生的事件。例如，一个场景可以定义和控制模拟的用户数、用户执行的操作以及用户运行其模拟时所用的计算机。

1. 创建场景

下面的内容是创建一个场景，用来模拟十个旅行代理同时登录系统、搜索航班、购买机票、查看路线和注销系统。

1）打开 Mercury LoadRunner

选择“开始”→“程序”→Mercury LoadRunner→LoadRunner。打开 Mercury LoadRunner Launcher 对话框。

2）打开 Controller

在“负载测试”选项卡中，单击“运行负载测试”。将打开 LoadRunner Controller。默认情况下，Controller 打开时将显示“新建场景”对话框，如图 12.10 所示。

3）选择场景类型

选择“手动场景”。Controller 允许用户选择各种不同的场景类型。

4）向负载测试添加脚本

单击“浏览”，找到<LoadRunner 安装文件夹>\Tutorial 目录中的 basic_script。“可用脚本”部分和“场景中的脚本”部分中将显示该脚本。单击“确定”按钮。LoadRunner Controller 的“设计”选项卡中将显示您创建的场景。

2. 设计场景

Controller 窗口的“设计”选项卡包含“场景计划”和“场景组”两个主要部分，如图 12.11 所示。

（1）场景计划：在“场景计划”部分，用户可以设置负载行为以准确描绘用户行为。

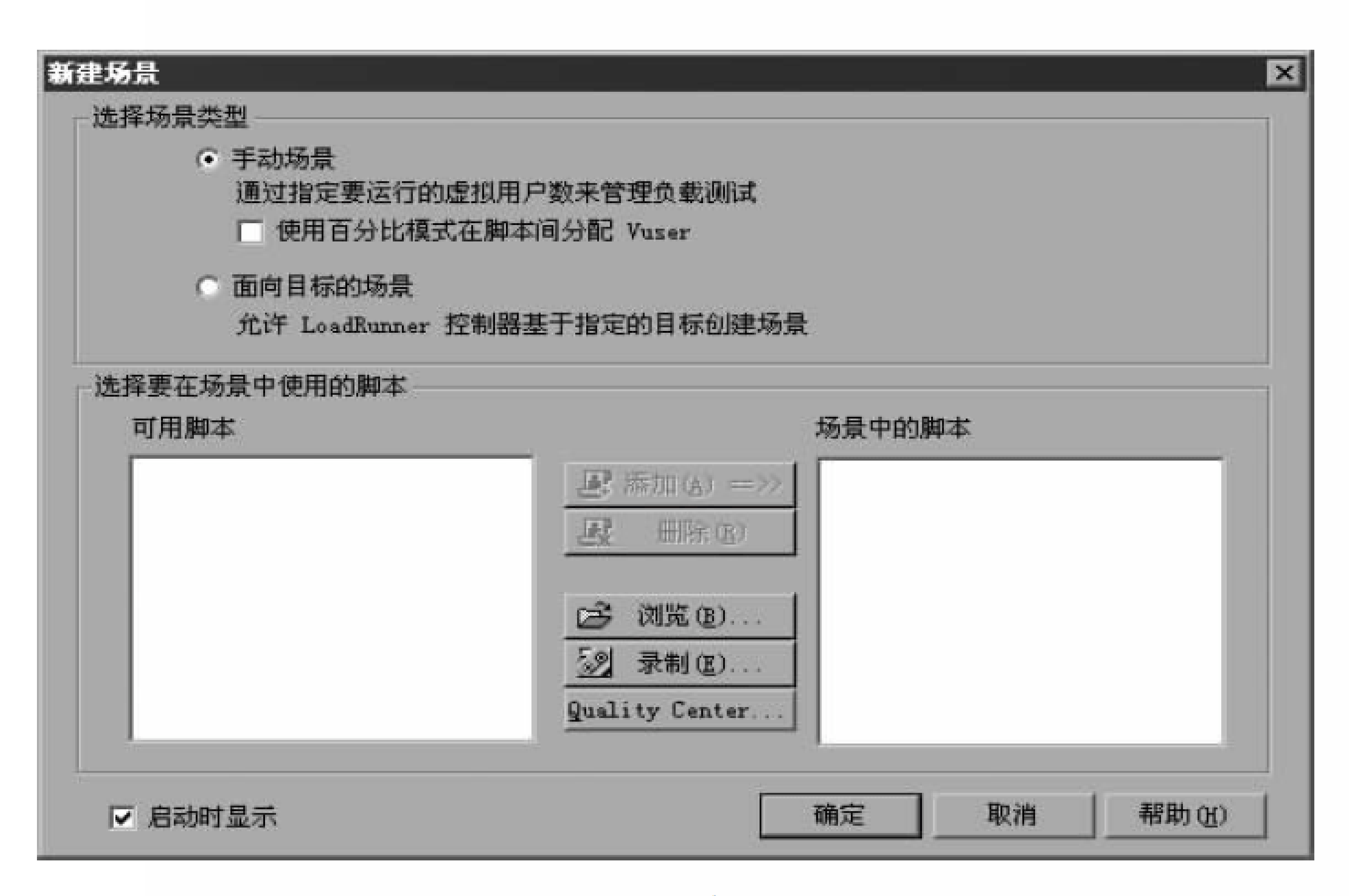


图 12.10　新建场景类型

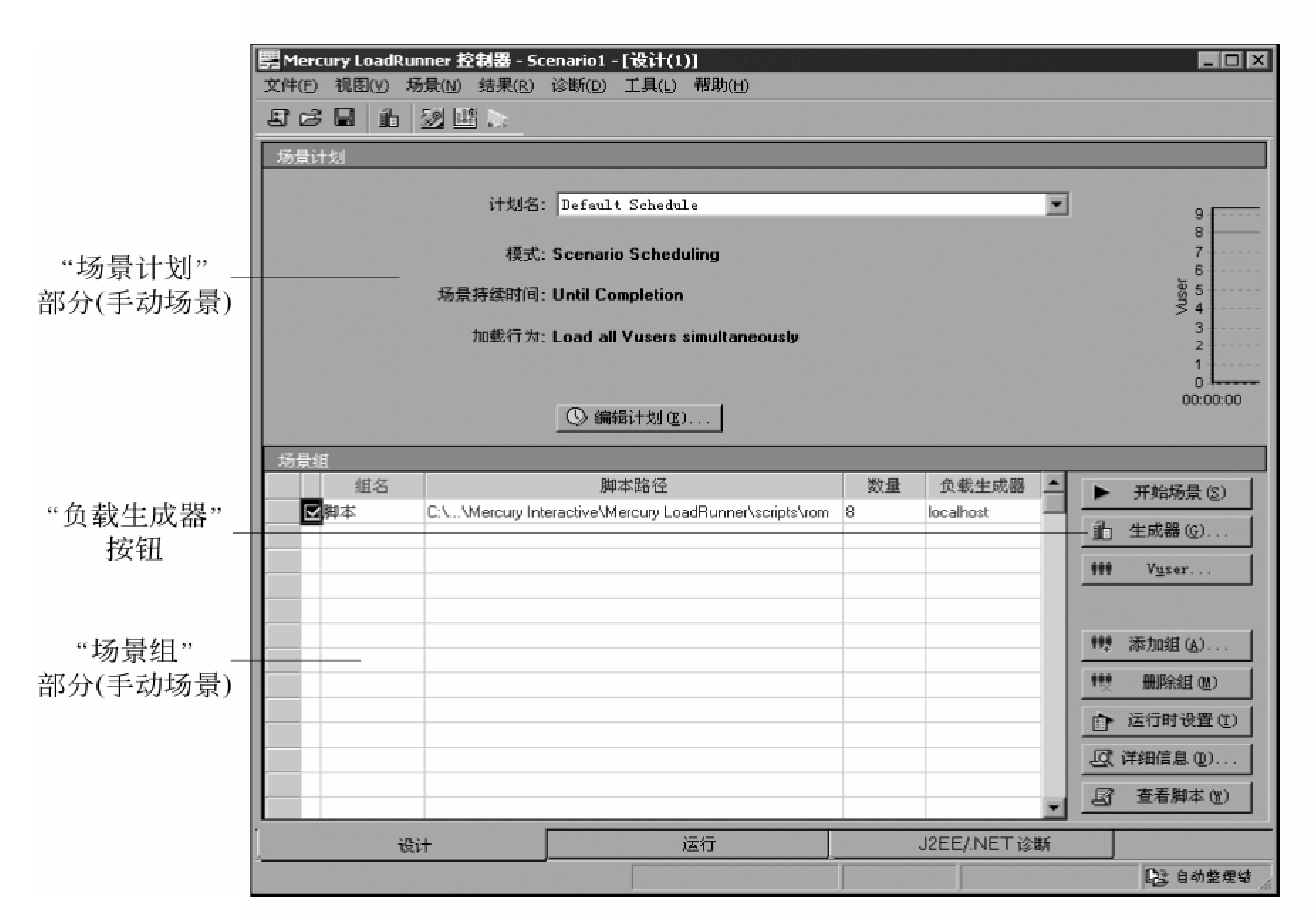


图 12.11　Controller 窗口的"设计"选项卡

用户可以确定将负载应用于应用程序的频率、负载测试持续时间和停止负载的方式。

（2）场景组：可以在"场景组"部分配置 Vuser 组。用户可以创建不同组来代表系统的典型用户。用户可以定义这些典型用户运行的操作、运行的 Vuser 数以及 Vuser 运行时所用的计算机。

（3）负载生成器：负载生成器是通过运行 Vuser 在应用程序中创建多个虚拟用户，

从而模拟多个用户进行操作。用户可以使用多台负载生成器计算机,并在每台计算机上创建许多个虚拟用户。

3. 计划场景

由于通常不会有多个典型用户恰好同时登录和注销系统,因此,LoadRunner 的 Controller 计划生成器允许用户建立较准确描绘典型用户行为的场景计划。例如,在创建手动场景后,设置场景的持续时间或选择在场景中逐渐运行和停止 Vuser。

下面,使用 Controller 计划生成器更改默认负载设置。

1) 更改场景计划默认设置

单击"编辑计划"按钮,将打开计划生成器,如图 12.12 所示。



图 12.12 计划生成器

2) 指定逐渐开始。

在"加压"选项卡中,将设置更改为"每 15 秒开始 2 个 Vuser"。

3) 计划持续时间。

在"持续时间"选项卡中,将设置更改为"在加压完成之后运行 3 分钟"。

4) 计划逐渐关闭。

在"减压"选项卡中,将设置更改为"每 30 秒停止 5 个 Vuser"。单击"确定"按钮。

12.3.3 使用 Controller 运行场景

完成了负载测试场景的设计,接下来运行该测试并观察应用程序如何在负载下运行。在开始运行测试之前,应该先熟悉 Controller 窗口的"运行"选项卡视图。"运行"选项卡是管理和监控测试的控制中心。

在"运行"选项卡中打开"运行"视图,如图 12.13 所示。

"运行"视图包含五个主要部分:

(1) 场景组:位于左上窗格中,可以查看场景组中的 Vuser 的状态。使用该窗格右侧的按钮可以启动、停止和重置场景,查看单个 Vuser 的状态,并且可以手动添加更多的

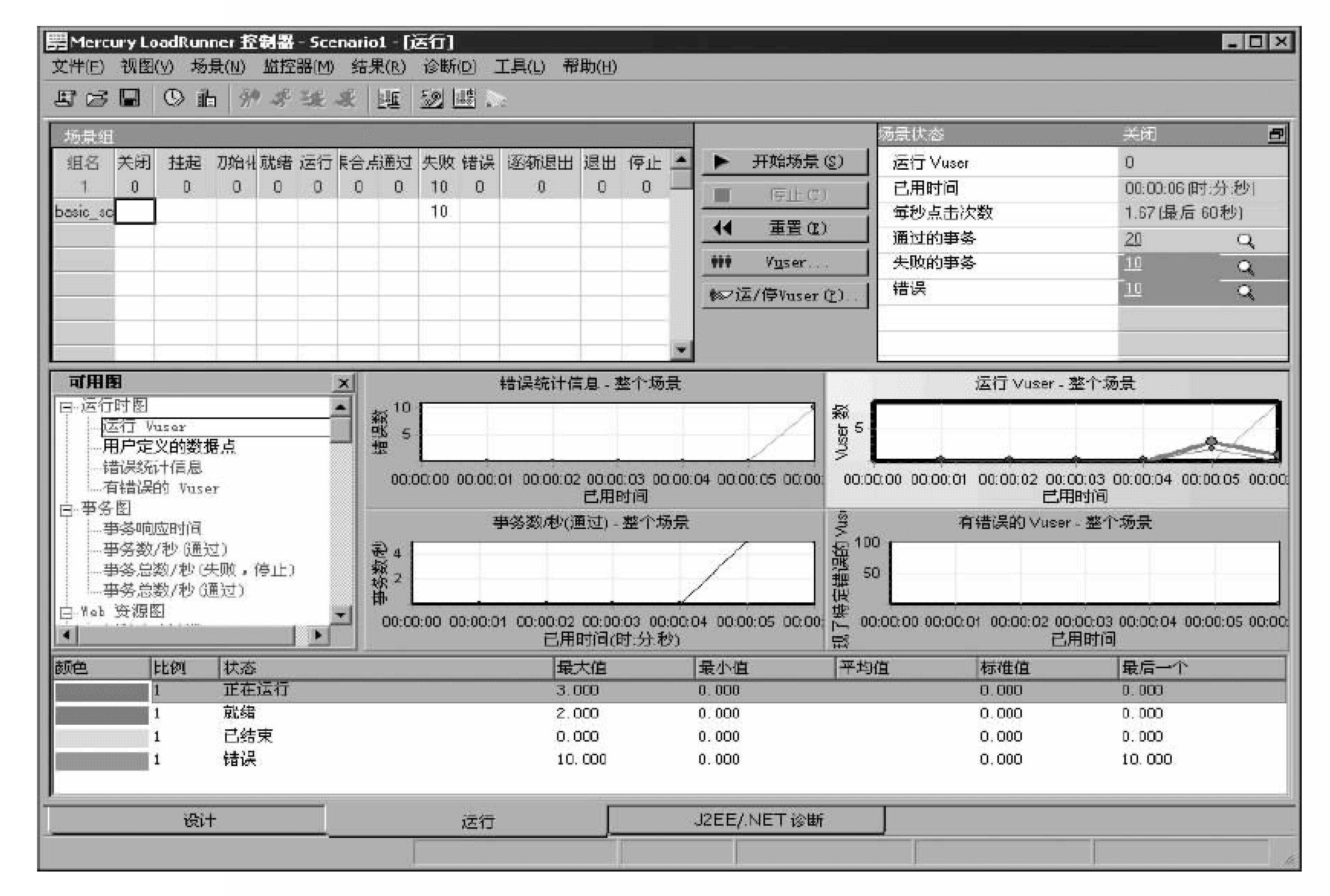


图 12.13　打开“运行”视图

Vuser，从而增加场景运行期间应用程序上的负载。

（2）场景状态：位于右上窗格中，可以查看负载测试的概要，其中包括正在运行的 Vuser 数以及每个 Vuser 操作的状态。

（3）可用图树：位于中部左侧窗格中，可以查看 LoadRunner 图列表。若要打开图，请在该树中选择一个图，然后将其拖动到图查看区域中。

（4）图查看区域：位于中部右侧窗格中，查看一至八个图（执行“视图”→“查看图”）。

（5）图例：位于底部窗格中，可以查看选定图中的数据。

运行步骤如下所示：

① 开始场景。单击“开始场景”按钮开始运行测试。Controller 开始运行场景。

② 通过 Controller 的联机图监控性能。测试运行时，通过 LoadRunner 的集成监控器查看应用程序如何实时执行以及潜在瓶颈所在位置，也可在 Controller 的联机图上查看监控器收集的性能数据。联机图显示在“运行”选项卡的图查看区域。

12.3.4　分析场景结果

现在已完成了场景运行，使用 LoadRunner Analysis 分析场景运行期间生成的性能数据，将性能数据收集到详细的图和报告中，从而可以轻松地确定和标识应用程序中的瓶颈以及提高系统性能所需的改进。

下面提供了一个 Analysis 会话示例，该会话所基于的场景与前面运行的场景相似。

（1）从 Controller 的菜单中选择“工具”→ Analysis 或选择“开始”→“程序”→ Mercury LoadRunner→“应用程序”→Analysis 来打开 LoadRunner Analysis。

（2）在 Analysis 窗口中，选择“文件”→“打开”。将打开“打开现有 Analysis 会话文

件”对话框。

(3) 在<LoadRunner 安装目录>\Tutorial 文件夹中，选择 analysis_session 并单击“打开”按钮。Analysis 将在 Analysis 窗口中打开该会话文件。

1. 概要报告

LoadRunner Analysis 打开时显示概要报告。概要报告提供有关场景运行的一般信息。在报告的统计信息概要中，可以了解到测试中运行的用户数，查看其他统计信息(如总/平均吞吐量和总/平均点击次数)。报告的事务概要列出了每个事务的行为概要。

2. 查看图

Analysis 窗口左窗格的图树中列出了已经打开可供查看的图。这些图显示在 Analysis 窗口右窗格的图查看区域中。

3. 平均事务响应时间

通过平均事务响应时间图，可以查看在场景运行的每一秒期间有问题的事务行为。

(1) 在图树中单击“平均事务响应时间”。平均事务响应时间图即显示在图查看区域中。

(2) 在图例中，单击 check_itinerary。check_itinerary 事务即突出显示在图中，如图 12.14 所示。

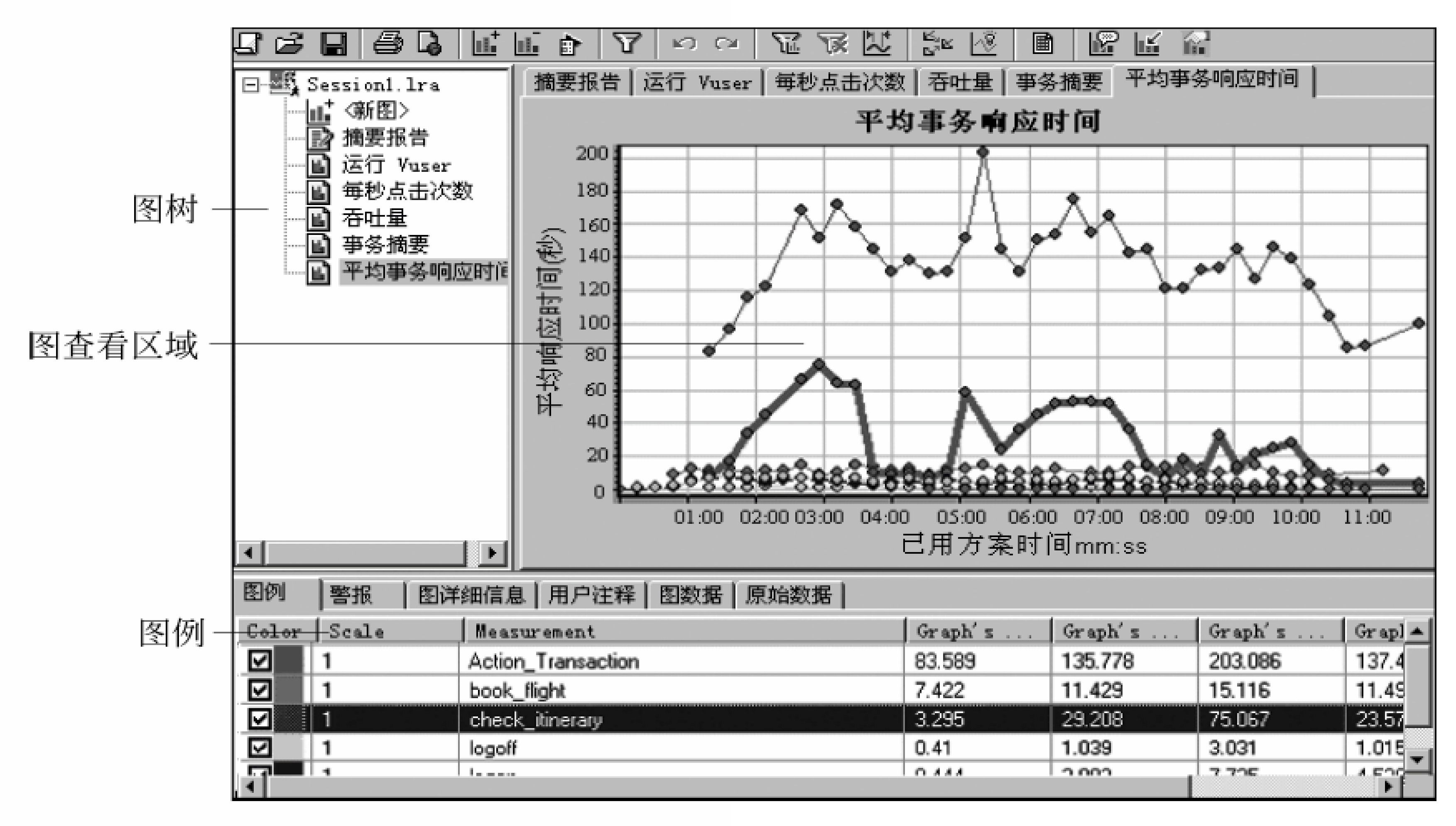


图 12.14 查看图表

4. 合并图和关联图

将两个图联系起来，就会看到一个图的数据会对另一个图的数据产生影响。这称为将两个图关联。例如，您可以将正在运行的 Vuser 图和平均事务响应时间图相关联，来

了解大量的 Vuser 对事务的平均响应时间产生的影响。

(1) 在图树中单击“正在运行的 Vuser”，查看正在运行的 Vuser 图。

(2) 右击正在运行的 Vuser 图并选择“合并图”。

(3) 在“选择要合并的图”列表中，选择“平均事务响应时间”。

(4) 在“选择合并类型”区域中，选择“关联”，然后单击“确定”按钮。

Vusers 和平均事务响应时间如图 12.15 所示。

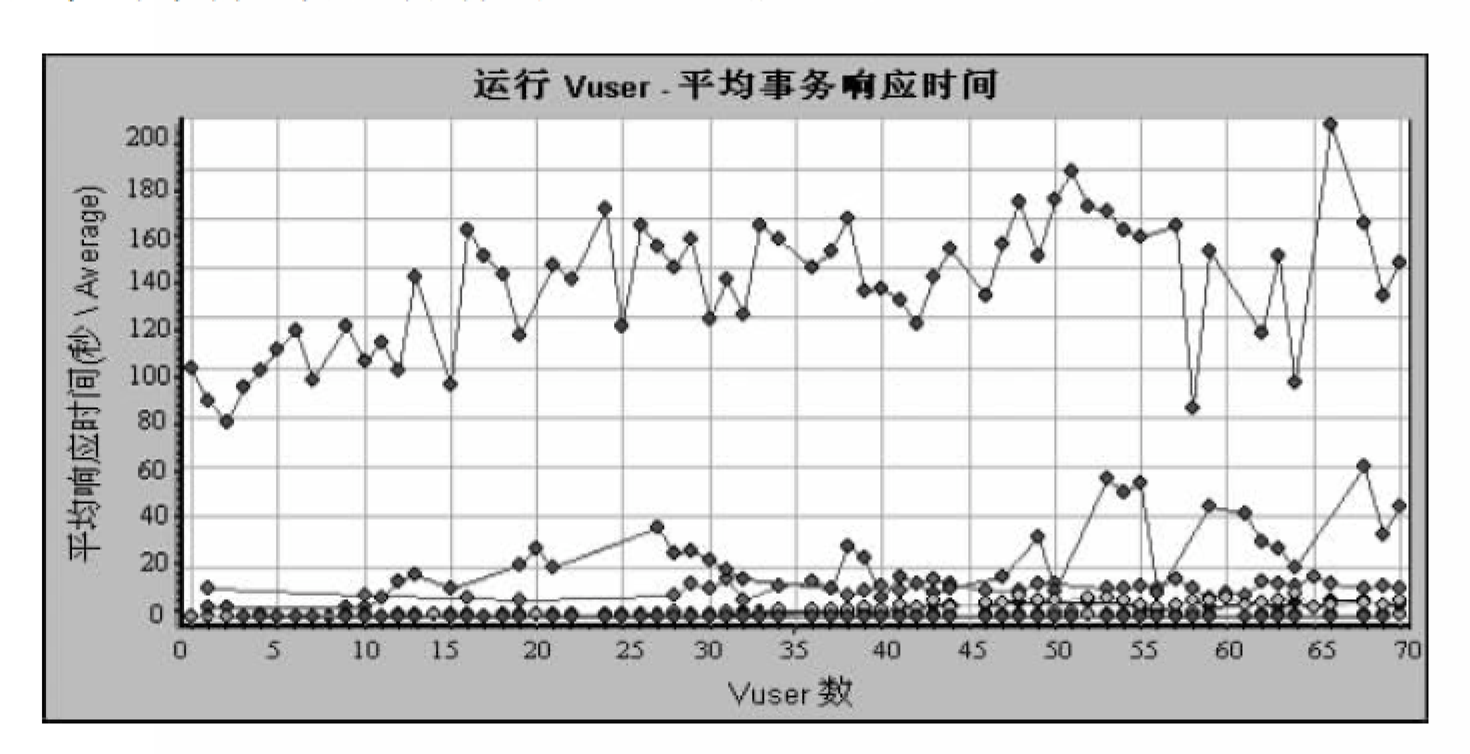


图 12.15 Vusers 和平均事务响应时间

另一个 Analysis 工具自动关联用来合并所有包含可能已对给定事务产生影响的数据的图，事务与每个元素的关联都显示出来。

5. 筛选图数据和排序图数据

对图数据进行筛选，显示特定场景段的较少事务。对图数据进行排序，以更多相关方式来显示数据。

(1) 在图树中单击“平均事务响应时间”打开该图。

(2) 右击该图并选择“设置筛选器 / 分组方式”。

(3) 在“事务名称”值框中，选择 check_itinerary 并单击“确定”按钮。筛选的图仅显示 check_itinerary 事务并隐藏所有其他事务。

6. 发布 HTML 报告和 Microsoft Word 报告

采用 HTML 报告或 Microsoft Word 报告的形式发布 Analysis 会话的结果。



缺陷管理工具

作为测试工作的重要功能之一，缺陷跟踪管理确保发现的缺陷能够及时得到处理，必须具有良好的工具支持，本章介绍两种广泛应用的缺陷跟踪管理工具：Bugzilla 和 JIRA。

13.1 Bugzilla

Bugzilla 是 Mozilla 公司提供的开源的缺陷跟踪工具，能够为软件组织建立一个完善的缺陷跟踪体系，包括报告缺陷、查询缺陷记录并产生报表、处理解决缺陷、管理员系统初始化和设置等。Bugzilla 从网站 http://www.bugzilla.org/ 下载，需配置数据库系统 MySQL、Web 服务器(如 Tomcat)，在 UNIX/Linux 和 Windows 系统上运行。

Bugzilla 具有以下特点：

(1) 基于 Web 方式，安装简单、运行方便快捷、管理安全。

(2) 提供强大的查询功能，根据各种条件组合进行组合查询。

(3) 缺陷从产生到解决，都有详细的操作记录，允许用户随时获取缺陷的历史记录。

(4) 具有强大的基于数据库的报表生成功能。

(5) 具有灵活和完善的权限管理功能，管理员可以根据需要定义由个人或者小组构成的访问组。

(6) 模型化的验证模块，使用户方便地添加所需系统验证。

(7) 管理员可以根据用户所在地域而配置使用本地的字体进行页面显示。

(8) 评论回复连接：对缺陷的评论提供直接的页面连接，帮助复查人员评审缺陷。

Bugzilla 的缺陷处理流程如下所示：

(1) 测试人员或开发人员发现缺陷后，判断属于哪个模块的问题，填写缺陷报告后，通过邮件通知项目组长或直接通知开发者。

(2) 项目组长根据具体情况，重新分配给缺陷所属的开发者。

(3) 开发者收到邮箱信息后，判断是否为自己的修改范围。

- 若不是，重新分配给项目组长或应该负责的开发者。
- 若是，进行处理并给出解决方法。

(4) 测试人员查询开发者已修改的缺陷，进行重新测试。

- 经验证无误后，修改状态为已证实。待整个产品发布后，修改为关闭。
- 若还有问题，重新打开，状态重新变为新，并发邮件通知相关人员。

(5) 如果这个缺陷一周内没被处理过，Bugzilla 就会通过邮件通知它的属主，直到缺陷被处理。

Bugzilla 操作流程总共有如下 7 步骤，分别是新建账号、报告缺陷、处理缺陷、验证已修改的缺陷、确认缺陷是否存在、查找缺陷和报表的生成。下面依次进行详细的讲解。

1. 新建账号

(1) 单击 Open a new Bugzilla account 链接，输入用户的 E-mail 地址，单击 Create Account。

(2) 用户会收到一封电子邮件，邮件中包含用户的登录账号和口令，这个口令是 Bugzilla 系统随机生成的，可以根据需要进行变更。

2. 报告缺陷

报告一个新的缺陷时，需在图 13.1 所示的界面中输入缺陷的相关信息。

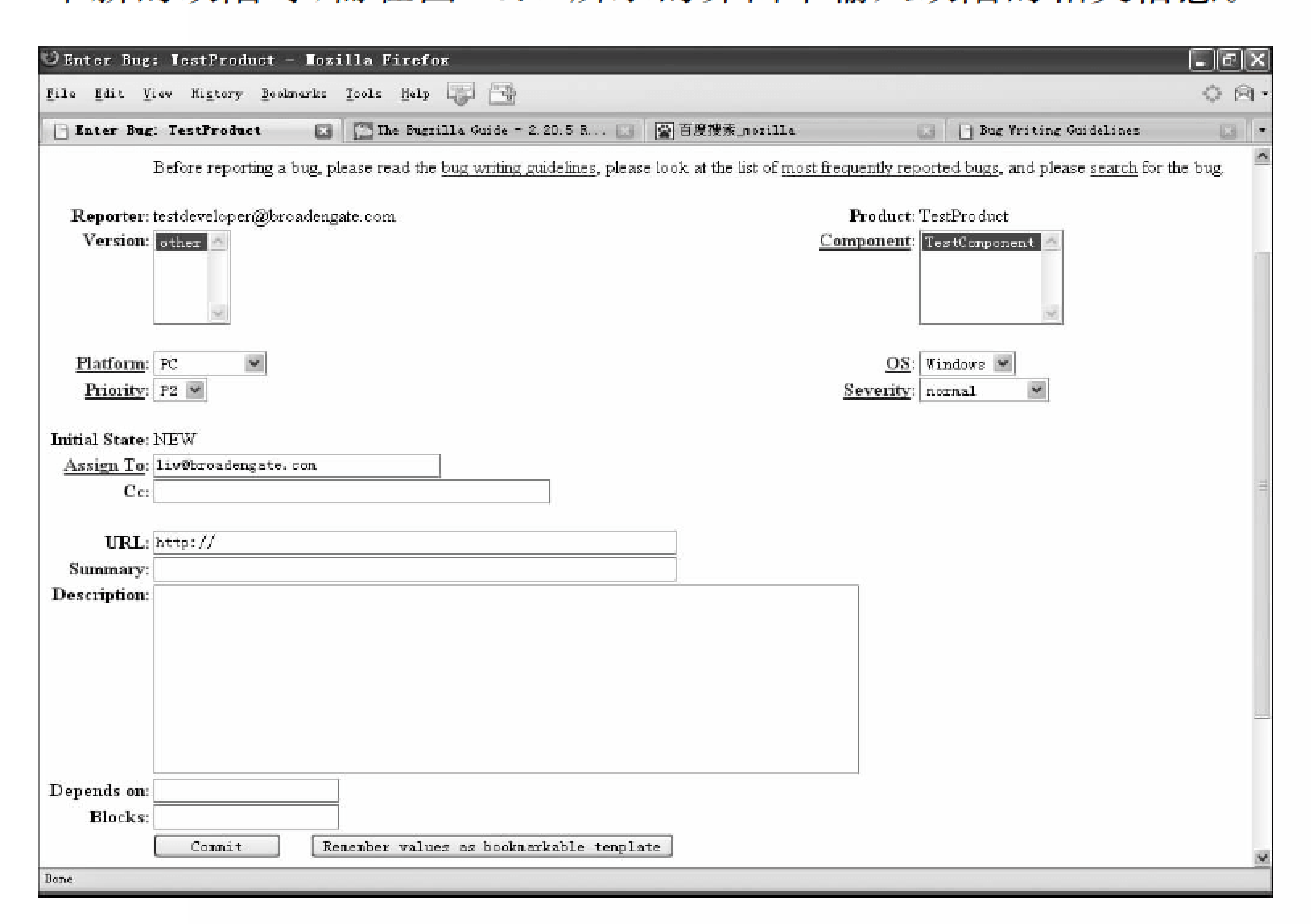


图 13.1 报告一个新的 Bug

在输入的信息中，Priority 指缺陷的优先级。在 Bug 的处理流程中，有以下几种状态：

- Unconfirmed：待确认的。
- New：新提交的。
- Assigned：已分配的，指已分配给相关人员进行修复。
- Reopened：因问题未解决而重新打开的。
- Resolved：缺陷已修复而待返测的。
- Verified：已经过返测而待归档的。

- Closed：已归档的。

下面给出界面上参数的解释。

- Assign To 指将缺陷分配给哪一个人员进行处理。
- Cc 指可同时接到 Bug 报告通知的人员，如果为多人，需用“，”隔开。
- URL 是一个超链接地址，引导处理人找到与缺陷报告相关联的信息。
- Summary 是概述部分，保证处理人在阅读时能够清楚提交者在进行什么操作的时候发现了什么问题。如果是通用组件部分的测试，则必须将这一通用组件对应的功能名称写入概述中，以便今后查询。

在 Description 中要详细说明下列情况：

- Depends on 是指如果该 Bug 必须在其他 Bug 修改以后才能够修改，则填写 Bug 的编号。
- Blocks 是指如果该 Bug 影响其他 Bug 的修改，则在此项目后填写被影响的 Bug 编号。
- Component 是指 Bug 所在的软件模块。

Severity 指缺陷的严重程度，可选择以下选项：

- Blocker，阻碍开发和/或测试工作。
- Critical，关键性缺陷，如死机，丢失数据，内存溢出等。
- Major，较大的功能缺陷。
- Normal，普通的功能缺陷。
- Minor，较轻的功能缺陷。
- Trivial，产品外观上的问题或不影响使用的毛病。

填写了以上信息后，单击 Commit 按钮，提交 Bug，Bugzilla 会自动发送邮件通知负责处理缺陷的人员。

3. 处理缺陷

Bug 的属主(负责修复缺陷的人员)进行缺陷修复，操作方法如下：

登入 Bugzilla 系统后，单击浏览器下方的 Saved Searches：My Bugs 按钮进入 Bug 管理界面，选择要修复的 Bug，修复完毕后，给出解决方式并填写 Additional Comments，如图 13.2 所示。

Bug 的解决方式有以下几种：

- FIXED：问题已经修复。
- INVALID：描述的问题不是一个 Bug。
- WONTFIX：描述的问题将永远不会被修复。
- LATER：描述的问题将不会在产品的这个版本中解决。
- DUPLICATE：描述的问题与以前的某个 Bug 重复。
- WORKSFORME：无法重现 Bug。

如果负责处理 Bug 的人员发现此 Bug 不属于自己的职责范围，可通知项目组长或测试人员重新分配此 Bug。

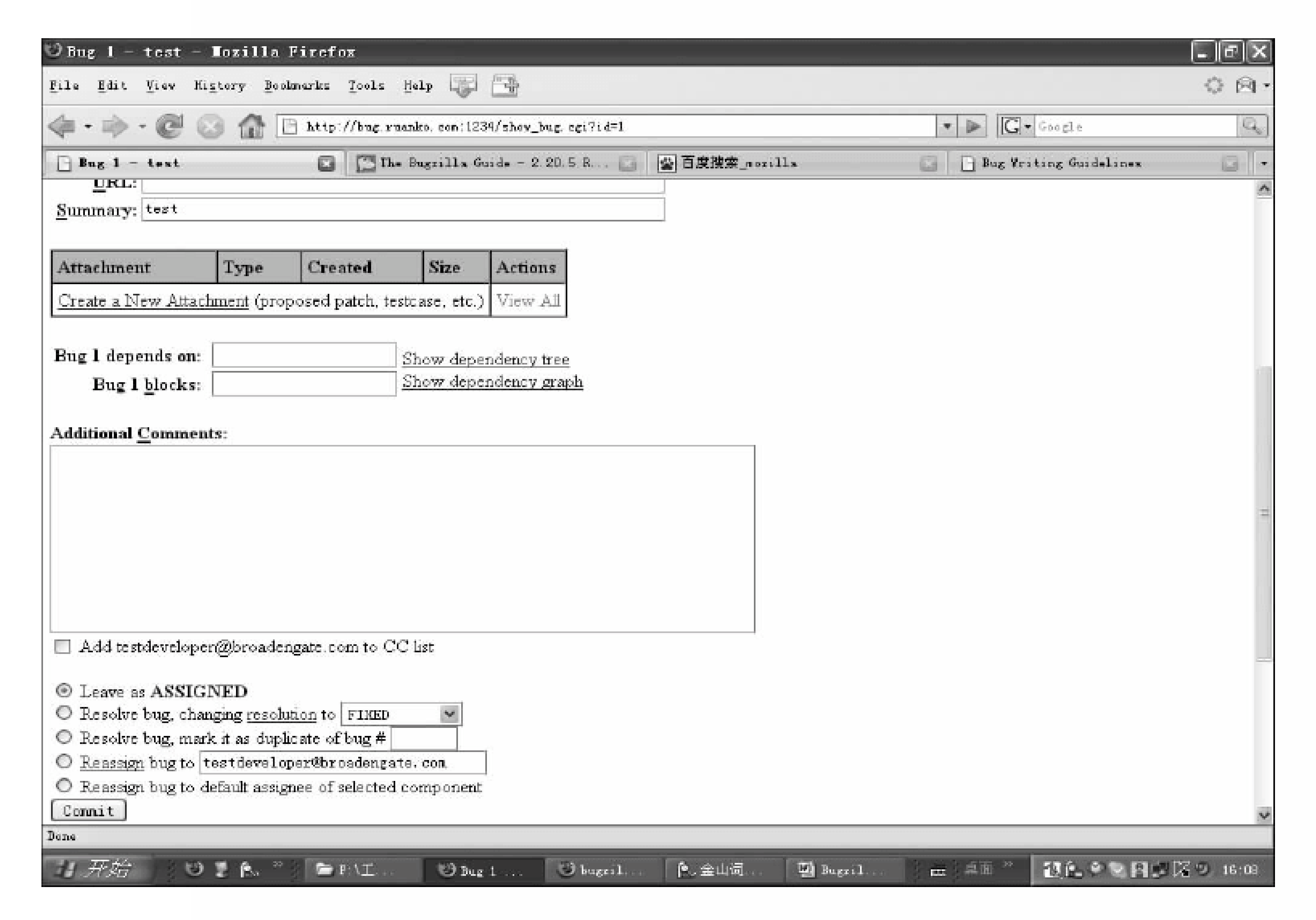


图 13.2　处理缺陷

4. 验证已修改的缺陷

开发人员处理完 Bug 并提交之后，测试人员查询开发者已修改的缺陷，即 Status 为 Resolved，Resolution 为 Fixed 的 Bug，进行回归测试。经验证无误后，将缺陷的状态改为 Verified。待整个产品发布后，修改为 Closed。若还有问题，Reopened，状态重新变为 New，并发邮件通知，如图 13.3 所示。

5. 确认缺陷是否存在

(1) 查询状态为 Unconfirmed 的 Bug。

(2) 测试人员对开发人员提交的 Bug 进行确认，确认 Bug 存在。

具体操作：选中 Confirm bug(change status to New)后，进行 commit。操作结果为 Bug 的状态变为 New。

6. 查找缺陷

使用 Bugzilla 的查询页面查找到系统中所有的 Bug 信息。Bug 报告中所有的字段信息都可作为查询条件，对于某些字段，可以选择多个值，在这种情况下，Bugzilla 会返回与任一值匹配的 Bug 记录。将一已执行的查询保存下来，成为一个“保存查询”(Saved Search)，显示在查询页面的页脚处，供以后重复使用。

7. 报表的生成

当使用查询功能得到 Bug 数据集后，可在此 Bug 数据集的基础上生成报表。

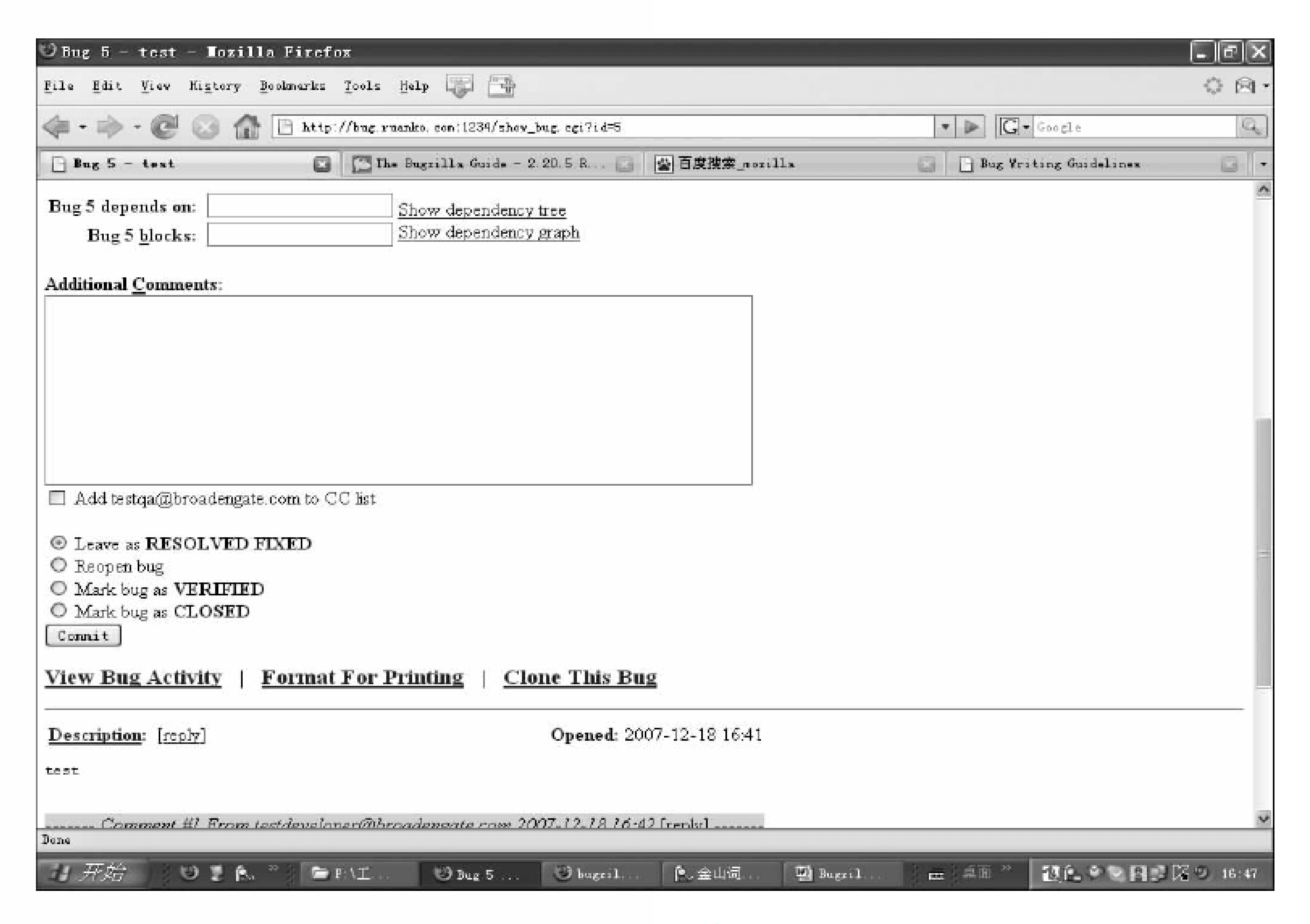


图 13.3　验证已修改的缺陷

Bugzilla 的报表分为两类：基于 HTML 表格的报表和基于图形的报表。其中基于图形的报表可绘制出线图、饼状图和柱状图。

例如，当使用查询功能查询出某一产品中的所有 Bug 记录后，可使用报表来显示出各构件中不同严重程度的 Bug 的分布状况，从而发现哪些模块的质量存在严重问题。当定义好报表参数后，单击 Generate Report，便可生成报表，且报表的形式可在 HTML 表格、线图、饼图和柱状图之间切换。

13.2　JIRA

JIRA 是澳大利亚 Atlassian 公司推出的一个问题跟踪管理软件，可跟踪和管理软件项目中出现的各种问题和缺陷。JIRA 非常灵活，提供各种系统设置功能，JIRA 可对以下各项提供定制功能：

- 问题信息字段：可灵活地定制描述问题的信息字段。
- 工作流：可定制问题跟中工作流。
- 权限分配方案：灵活地为用户组和用户分配操作权限。
- 邮件通知方案：定义当问题状态发生改变时，用邮件通知哪些用户/用户组。
- 操作界面：可自定义操作界面上的信息项和信息项的分布。

下面介绍 JIRA 的一些特点：

(1) JIRA 是“问题”管理工具，“问题”除包括缺陷外，还包括软件特征、任务和改进等。

(2) 具有高度可定制性，无论界面、缺陷跟踪工作流，还是问题属性字段，都可以由用

户进行定制，使系统具有高度的灵活性和适应性。

(3) 能够跟踪软件组件和版本。

(4) 强大查询功能，可将查询条件保存为过滤器，不同的用户可共享过滤器。

(5) 灵活的用户/用户组权限管理，可设置多个权限分配方案，不同的项目可以有不同的系统权限分配。

(6) 具有 E-mail 通知策略，不同的项目可设置不同的邮件通知方案。

(7) 易与其他系统实现集成，监听器和服务程序能够提供与现有系统的双向信息交换。具有良好的可扩展性，完整的 Java 应用程序接口允许用户通过编写代码直接与 JIRA 连接，从而可无限制地扩展 JIRA。

(8) 具有很好的平台兼容性与移植性。

13.2.1 跟踪操作

使用 JIRA 可以很方便地进行缺陷跟踪流程中的各种操作。

1. 报告缺陷

先选择要报告的问题类型，如图 13.4 所示。除 Bug 外，JIRA 还可对其他各种类型的问题进行跟踪，如需求变更，软件改进等。

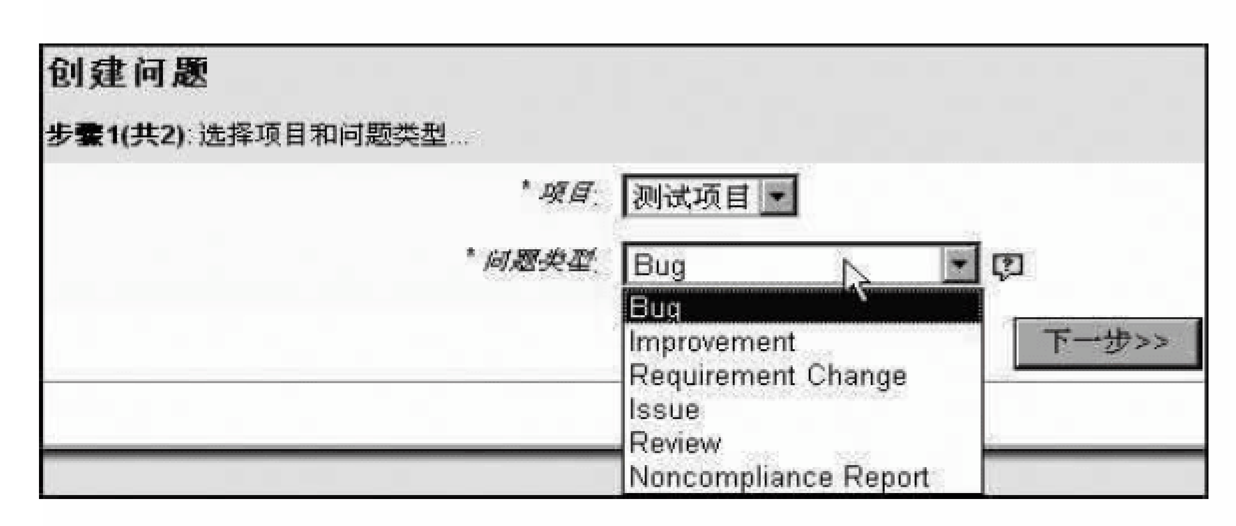


图 13.4 选择问题类型

单击“下一步”按钮，出现 Bug 信息填写页面，如图 13.5 所示，在该页面中填写 Bug 的详细信息。

以下简要介绍图 13.5 中各信息字段的含义。

- 摘要：简明扼要地描述缺陷。
- 优先级：分为危急、严重、一般、次要、轻微等级别。
- 组件：选择缺陷所在的项目中的组件。
- 受影响版本：当前出问题的版本。
- 解决版本：规划在哪一个版本解决 Bug，一般为出问题的版本。
- 分配给：选择分配给特定的人员进行处理，如果不指定，则自动分配。
- 环境：例如操作系统，软件信息，硬件规格等信息。
- 描述：对缺陷的详细描述。可附上出问题的 URL 地址，以方便追查故障。

2. 处理 Bug

开发人员首先查询分配给自己处理的缺陷，如图 13.6 所示。



图 13.5　填写 Bug 的详细信息

接受缺陷处理任务，准备开始处理缺陷，如图 13.7 所示。

处理完缺陷后，准备填写处理情况，如图 13.8 所示。

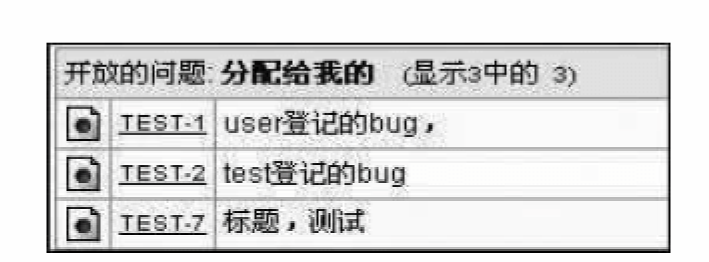


图 13.6　查询分配给自己处理的缺陷

可选工作流程
接受本Bug
操作
上传附件 到此问题
上传屏幕截图 到此问题

图 13.7　接受 Bug 处理任务

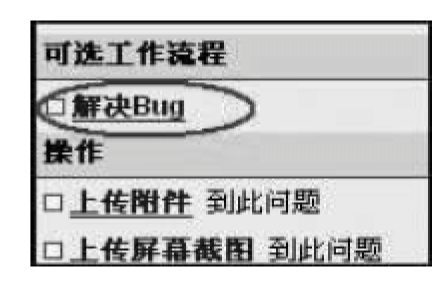


图 13.8　解决 Bug

填写缺陷处理情况，如图 13.9 所示。

缺陷的处理方式有如下多种情况：

- Fixed：已修复。
- Later：在以后的版本中修复。
- Invalid：描述的问题不是一个 Bug。
- Won't Fix：该 Bug 将不会被修复。
- Duplicate：描述的问题与以前的某个 Bug 重复。
- Cannot Reproduce：不能重现该 Bug。

3. 关闭或重新打开 Bug

对于已经处理的缺陷，可将其关闭，若缺陷仍存在问题，可将其重新打开(对于已经关闭的缺陷，也可将其重新打开)，如图 13.10 所示。

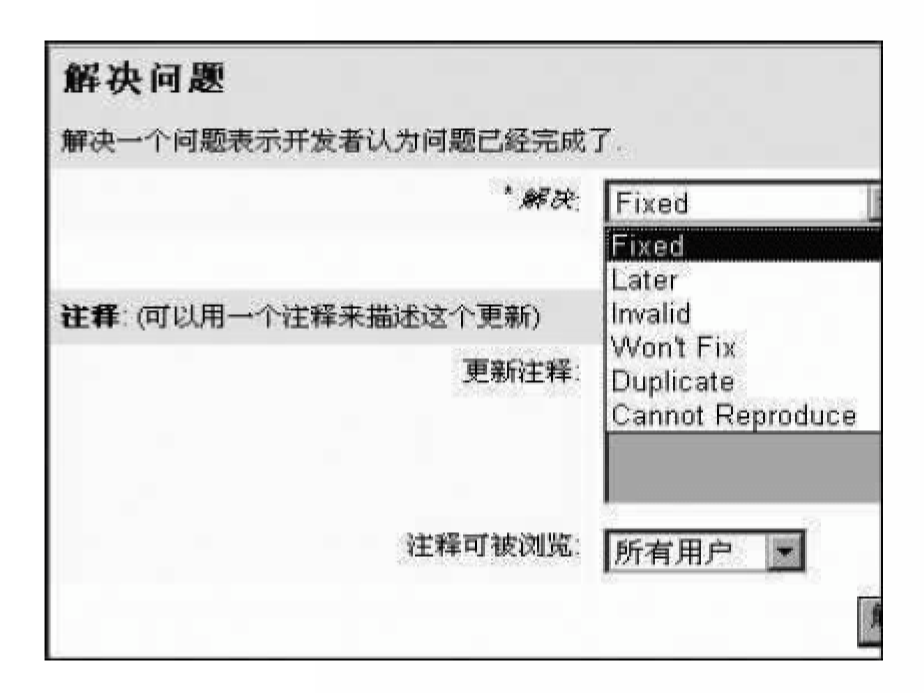


图 13.9　填写缺陷处理情况



图 13.10　关闭或重新打开 Bug

13.2.2　查询操作

JIRA 提供了很强的问题查询功能。在 JIRA 的主菜单中单击 Find Issues，可打开 JIRA 的问题浏览界面，在该界面左侧是问题查询表单，如图 13.11 所示。

图 13.11 显示输入或选择各种查询条件，例如属于某项目的缺陷、描述字段包含了某一字符串的缺陷、某人员报告的所有缺陷，等等。查询条件输入完毕后，单击 View 按钮，显示出所有满足条件的问题记录。

如果某一组查询条件可能会重复使用，可将其保存为“过滤器”。图 13.12 显示了 JIRA 的过滤界面。

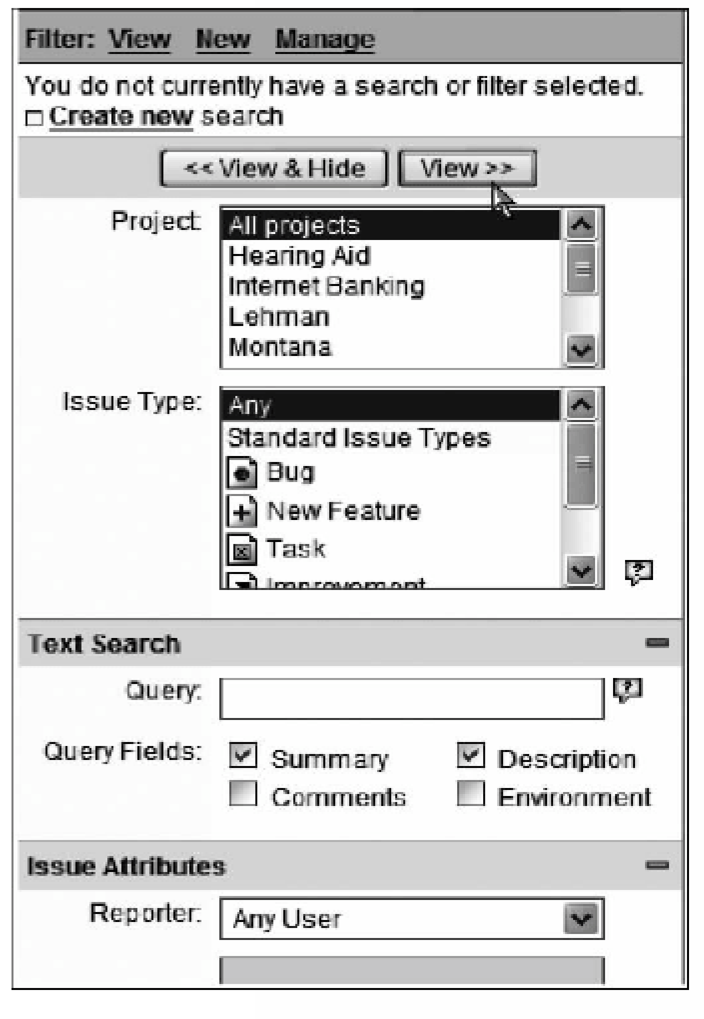


图 13.11　JIRA 的问题查询界面

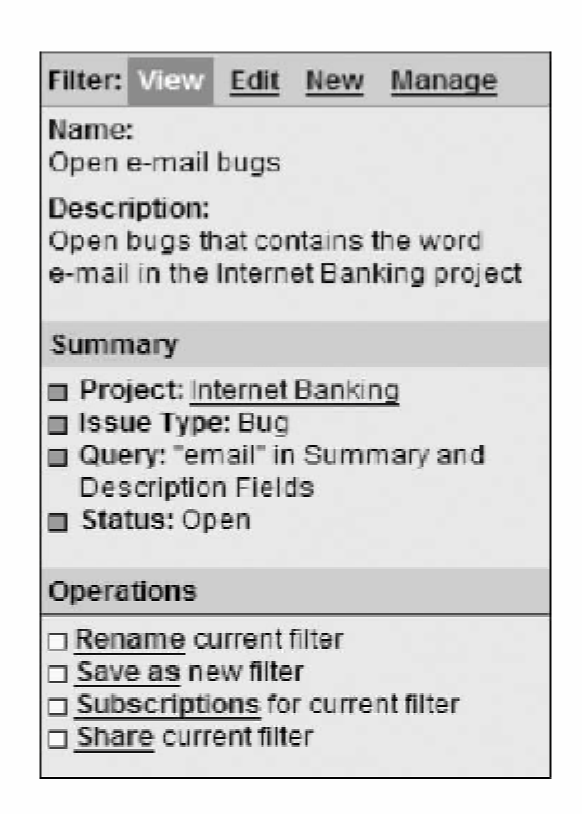


图 13.12　过滤器界面

在过滤器界面中，View 选项卡用于查看当前的过滤器。该选项卡中显示了过滤器的名称、描述、摘要、可用操作等信息。一个过滤器可以被重命名、保存为新的过滤器、订阅或共享。Edit 选项卡用于编辑当前过滤器的查询条件。New 选项卡用于创建一个新的

过滤器。Manage 选项卡用于管理所有已存在的过滤器。

13.2.3 生成报表

JIRA 可产生各种各样的报表，用于显示 JIRA 系统中各项数据统计结果。JIRA 内置的报表包括：路线图、变更日志、受关注问题、发布记录、时间跟踪报表、开发者工作量报表、版本工作量报表、单级分组报表。

1. 路线图报表

JIRA 为每个项目提供了一个路线图，显示出在将要发布的三个版本中需要解决的问题。路线图报表提供了对版本发布进展情况的概览。单击 JIRA 主菜单的 Browse，在项目列表中选择一个项目，然后单击 Road Map 选项卡，便可显示出该项目的路线图报表。

2. 变更日志报表

变更日志报表显示了在某一项目最近的三个发布中所解决的问题，与路线图报表相反，变更日志报表提供了一个过去的视图，显示最近版本中已解决问题的概览。单击 JIRA 主菜单的 Browse，在项目列表中选择一个项目，然后单击 Change Log 选项卡，便可显示出该项目的变更日志报表。

3. 受关注问题报表

受关注问题报表显示了一个项目中未被解决的问题，这些问题是按它们受关注的程度排序的，而受关注程度是按它们的"投票数"衡量的。因此该报表只有在"投票"功能有效时才能使用。单击 JIRA 主菜单的 Browse，在项目列表中选择一个项目，然后单击 Popular Issues 选项卡，便可显示出该项目的受关注问题报表。

4. 时间跟踪报表

该报表显示了一个特定产品版本中的问题的时间跟踪信息。它显示出特定问题的起始和当前时间估计，以及它们是否超前或滞后于起始的计划。

按以下步骤产生一个时间跟踪报表：

① 在项目浏览器页面中选择一个项目。

② 单击"Time Tracking Report"，打开时间跟踪报表参数设置表单。

③ 填写必要的参数后，单击 Report 按钮，生成类似图 13.13 所示的时间跟踪报表。

5. 开发者工作量报表

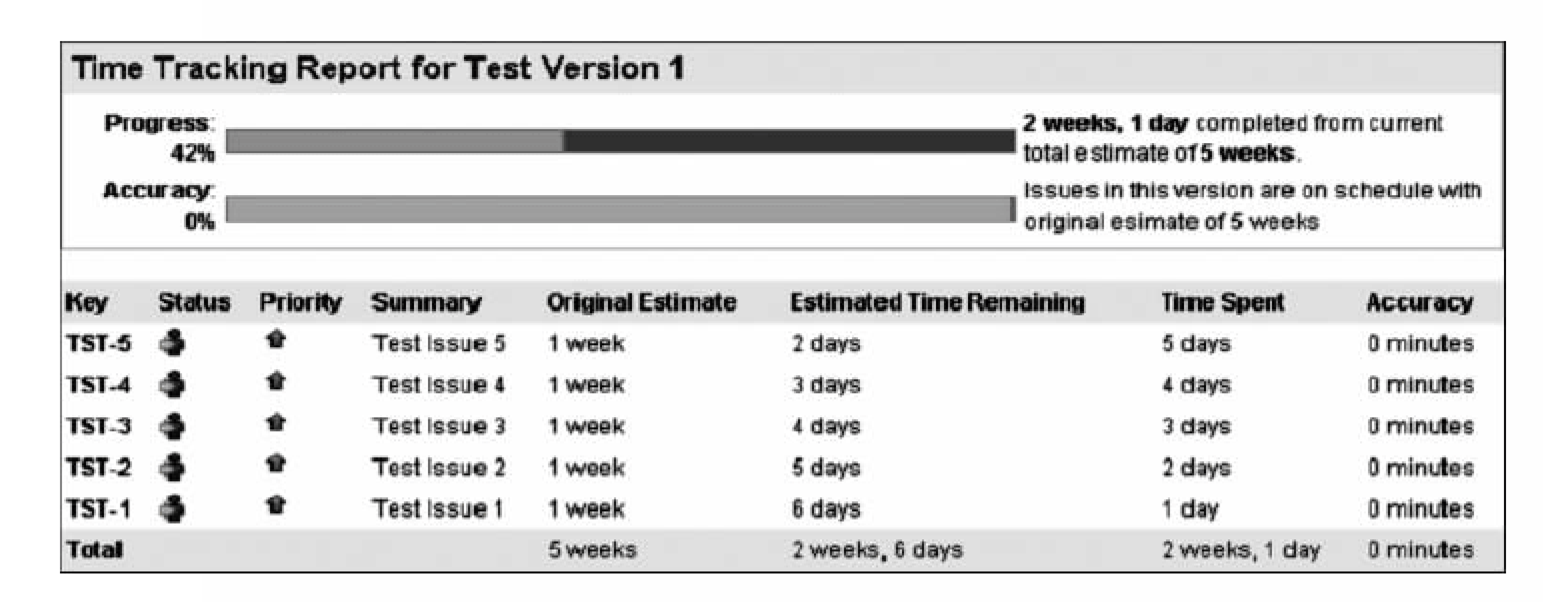
开发者工作量报表显示了分配给某一特定用户的问题的时间跟踪信息，它显示出未解决的分配给用户的问题数量和剩余的工作量。

按照如下步骤产生开发者工作量报表：

① 在项目浏览器页面中选择一个项目。

② 单击"Developer Workload Report"。

③ 选择一个用户。单击 Next 按钮，生成报表。



Time Tracking Report for Test Version 1

Progress: 42% — 2 weeks, 1 day completed from current total estimate of 5 weeks.

Accuracy: 0% — Issues in this version are on schedule with original esimate of 5 weeks

Key	Status	Priority	Summary	Original Estimate	Estimated Time Remaining	Time Spent	Accuracy
TST-5			Test Issue 5	1 week	2 days	5 days	0 minutes
TST-4			Test Issue 4	1 week	3 days	4 days	0 minutes
TST-3			Test Issue 3	1 week	4 days	3 days	0 minutes
TST-2			Test Issue 2	1 week	5 days	2 days	0 minutes
TST-1			Test Issue 1	1 week	6 days	1 day	0 minutes
Total				5 weeks	2 weeks, 6 days	2 weeks, 1 day	0 minutes

图 13.13 时间跟踪报表

6. 版本工作量报表

版本工作量报表显示了某项目中特定产品版本的当前工作量的时间跟踪信息。对于一个特定版本，该报表显示出未解决的分配给每个用户的问题数量和剩余工作量。

产生版本工作量报表的步骤如下：

① 在项目浏览器页面中选择一个项目。

② 单击 Version Workload Report。

③ 选择一个产品版本，单击 Next 按钮，生成报表。

7. 单级分组报表

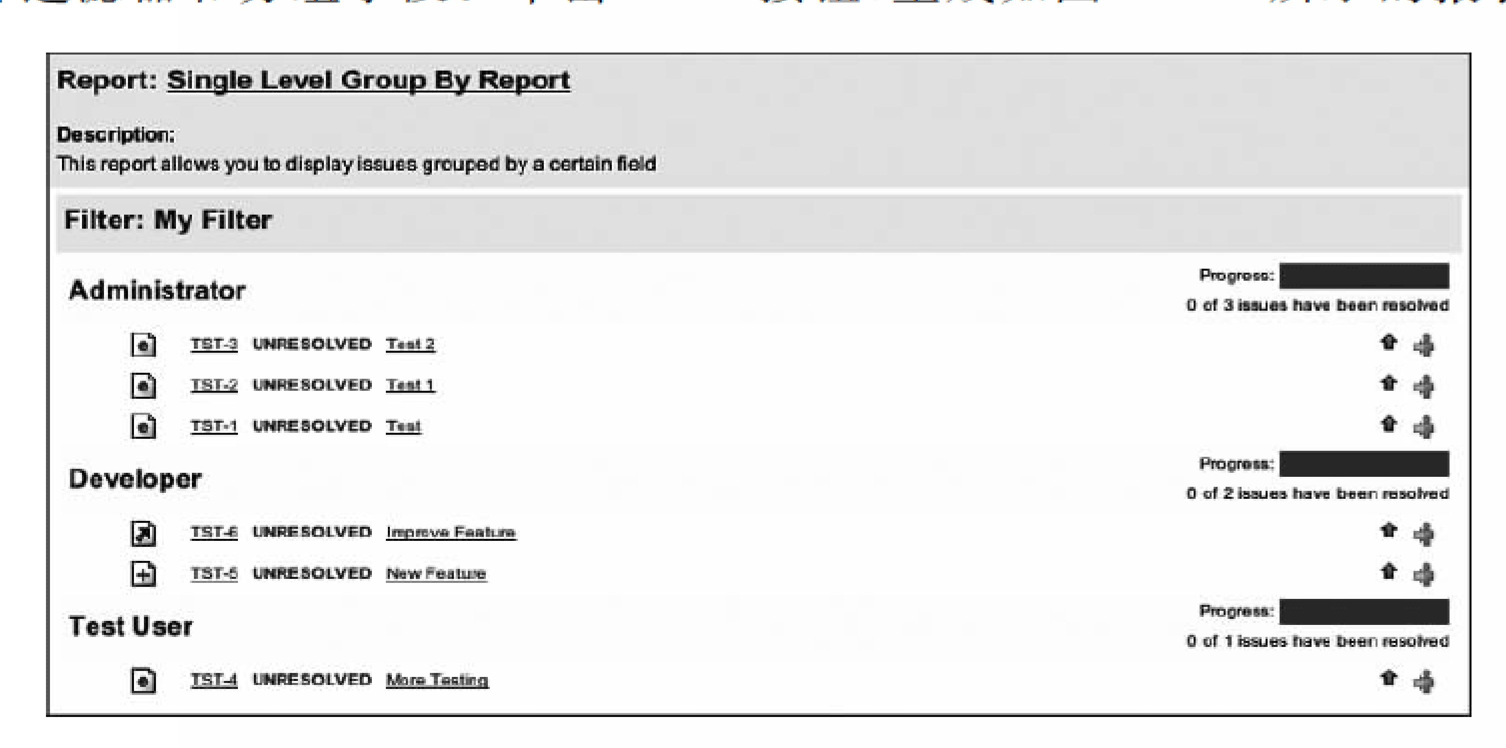
该报表显示出某一过滤器所查询到的所有问题，这些问题按照一个特定的字段分组。例如：一个过滤器用于查询某一特定版本中所有打开的 Bug，则可设计一单级分组报表，按照 Bug 处理人员分组显示这些 Bug。

产生版本工作量报表的步骤如下：

① 在项目浏览器页面中选择一个项目。

② 单击 Single Level Group by Report。

③ 选择过滤器和分组字段。单击 Next 按钮，生成如图 13.14 所示的报表。



Report: Single Level Group By Report

Description:
This report allows you to display issues grouped by a certain field

Filter: My Filter

Administrator — Progress: 0 of 3 issues have been resolved

TST-3 UNRESOLVED Test 2
TST-2 UNRESOLVED Test 1
TST-1 UNRESOLVED Test

Developer — Progress: 0 of 2 issues have been resolved

TST-6 UNRESOLVED Improve Feature
TST-5 UNRESOLVED New Feature

Test User — Progress: 0 of 1 issues have been resolved

TST-4 UNRESOLVED More Testing

图 13.14 单级分组报表



第14章 单元测试工具

单元测试工具中有 JUnit、C++ JUnit 等,针对不同的编程语言,本章介绍 JUnit 测试工具。

14.1 JUnit 特点

JUnit 是一个开源的 Java 测试框架,由 Erich Gamma 和 Kent Beck 开发,用于 Java 程序设计语言的类和函数的测试。JUnit 用于单元级测试,具有如下优势:

1. JUnit 是完全免费

JUnit 是完全开放源代码,在其基础上可以进行二次开发。

2. 使用方便

JUnit 可以快速地撰写测试并检测程序代码,JUnit 执行测试类似编译程序一样。

3. JUnit 检验结果并提供立即回馈

JUnit 自动执行并且检查结果,执行测试立即回馈信息。

4. JUnit 合成测试系列的层级架构

JUnit 引入了重构概念,它把测试组织成测试组,允许组合多个测试自动回归测试。

5. 与 IDE 的集成

与 Java 相关的 IDE 环境集成,形成测试及开发代码之间无缝连接。

14.2 JUnit 在 eclipse 中的使用

下面介绍 Eclipse 中使用 JUnit 进行测试的具体步骤:

运行 Eclipse。新建一个 workplace 项目,选择"文件"→"新建"→"项目",选择 Java 项目,单击"下一步"按钮。新建项目名 JUnit_Test,在 andycpp 包中编写 Calculator 类,实现简单的加、减、乘、除等计算功能,采用 JUnit4 进行测试 Calculator 类的各种方法。

① 在 eclipse 中编写 Calculator 类。为了进行测试，故意给出方法的错误代码。Calculator 类代码如下：

```
package andycpp;
public class Calculator {
    private static int result;                    //静态变量用于存储运行结果
    public void add(int n) {
        result=result+n;
    }
    public void substract(int n) {
        result=result-1;                  //Bug: 正确的应该是 result=result
-n
    }
    public void multiply(int n) {
    }                                     //此方法尚未写好
    public void divide(int n) {
          result = result/1;                 //Bug: 正 确 的 应 该 是 result =
result/n
    }
    public int getResult() {
        return result;
    }

}
```

② 将 JUnit4 单元测试包引入 JUnit_Test 项目。右击该项目，选择“Propertiers”，如图 14.1 所示。

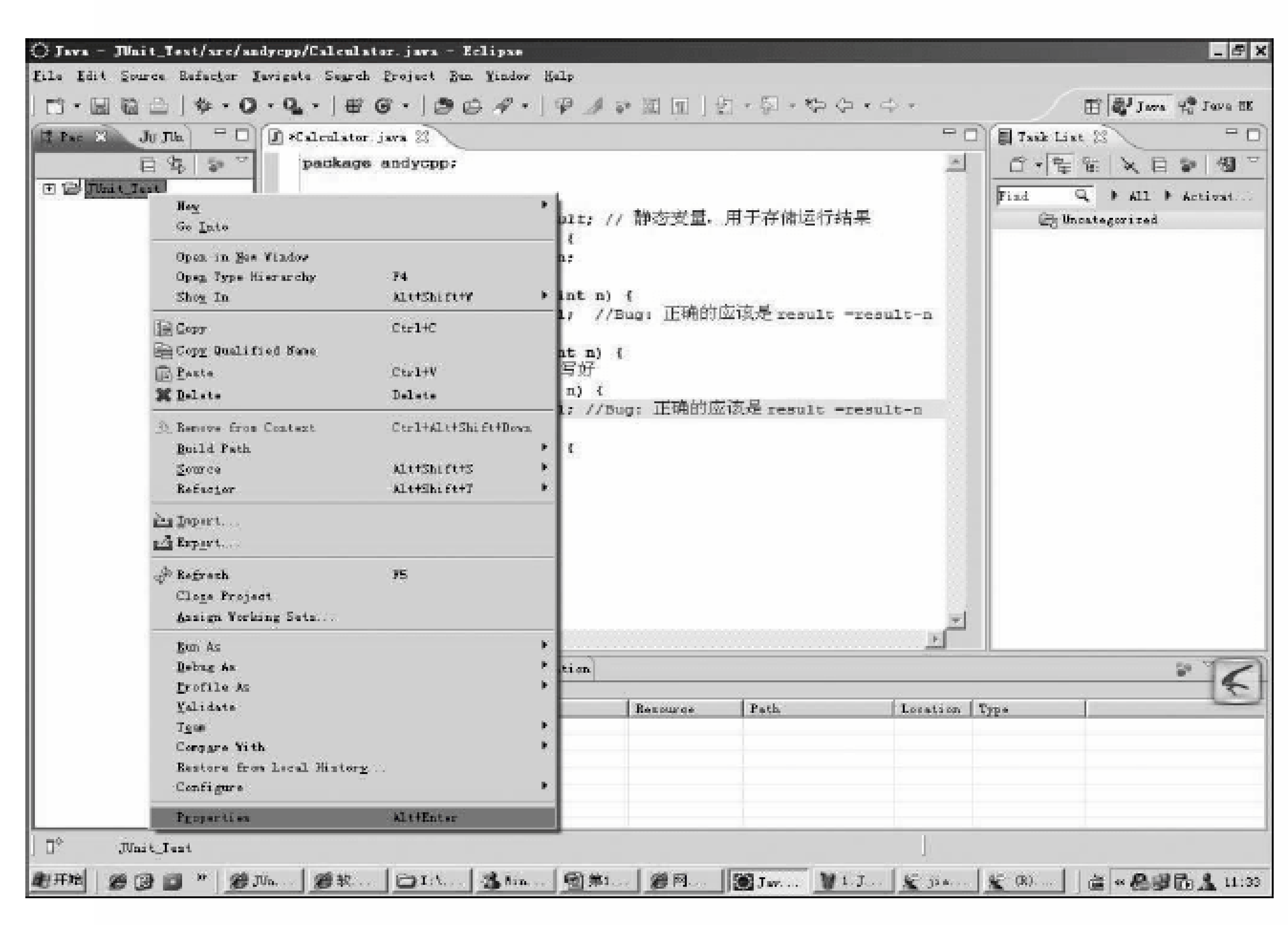


图 14.1　将 JUnit4 测试包引入 JUnit_Test 项目截图 1

在弹出的属性窗口中，首先在左边选择 Java Build Path，然后到右上选择 Libraries

标签，之后在最右边单击 Add Library 按钮，如图 14.2 所示。

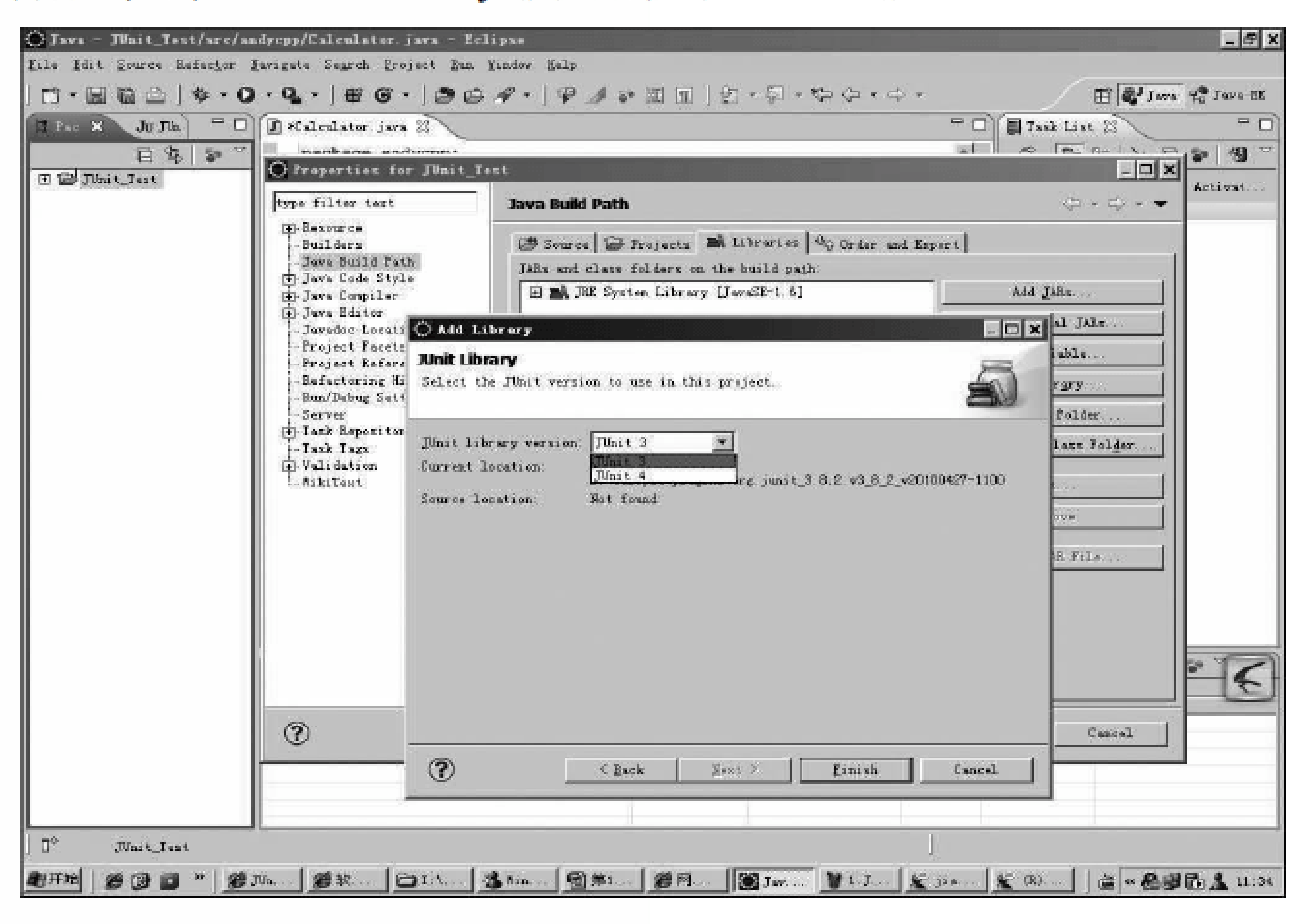


图 14.2　将 JUnit4 测试包引入 JUnit_Test 项目截图 2

在 14.2 图中选择 JUnit4 并单击“确定”按钮，将 JUnit4 软件包加入到 JUnit_Test 项目。

③ 生成 JUnit 测试框架。在 Eclipse 的 Package Explorer 中右击 Calculator 类，弹出快捷菜单，在其中选择 New a JUnit Test Case，如图 14.3 所示。

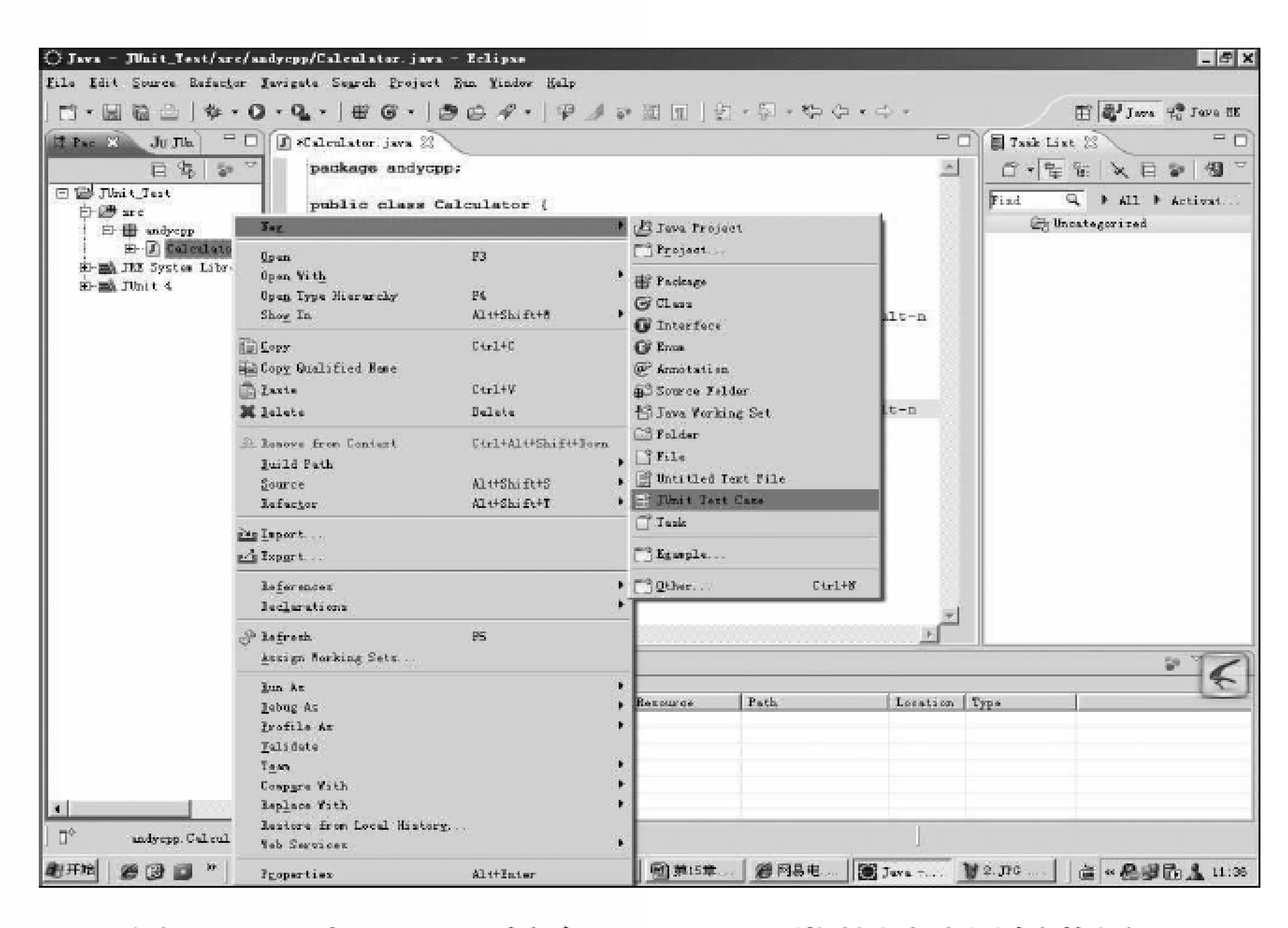


图 14.3　在 eclipse 创建 Calculator 类的测试用例截图 1

然后在弹出的对话框中选择 setUp()和 teardown()方法，如图 14.4 所示。

单击 Next 按钮后，系统会自动列出 Calculator 类中所包含的方法，如图 14.5 所示，选择所需测试的“加、减、乘、除”四个方法进行测试。

eclipse 自动生成名为 CalculatorTest 新类，代码中包含一些空的测试用例。将

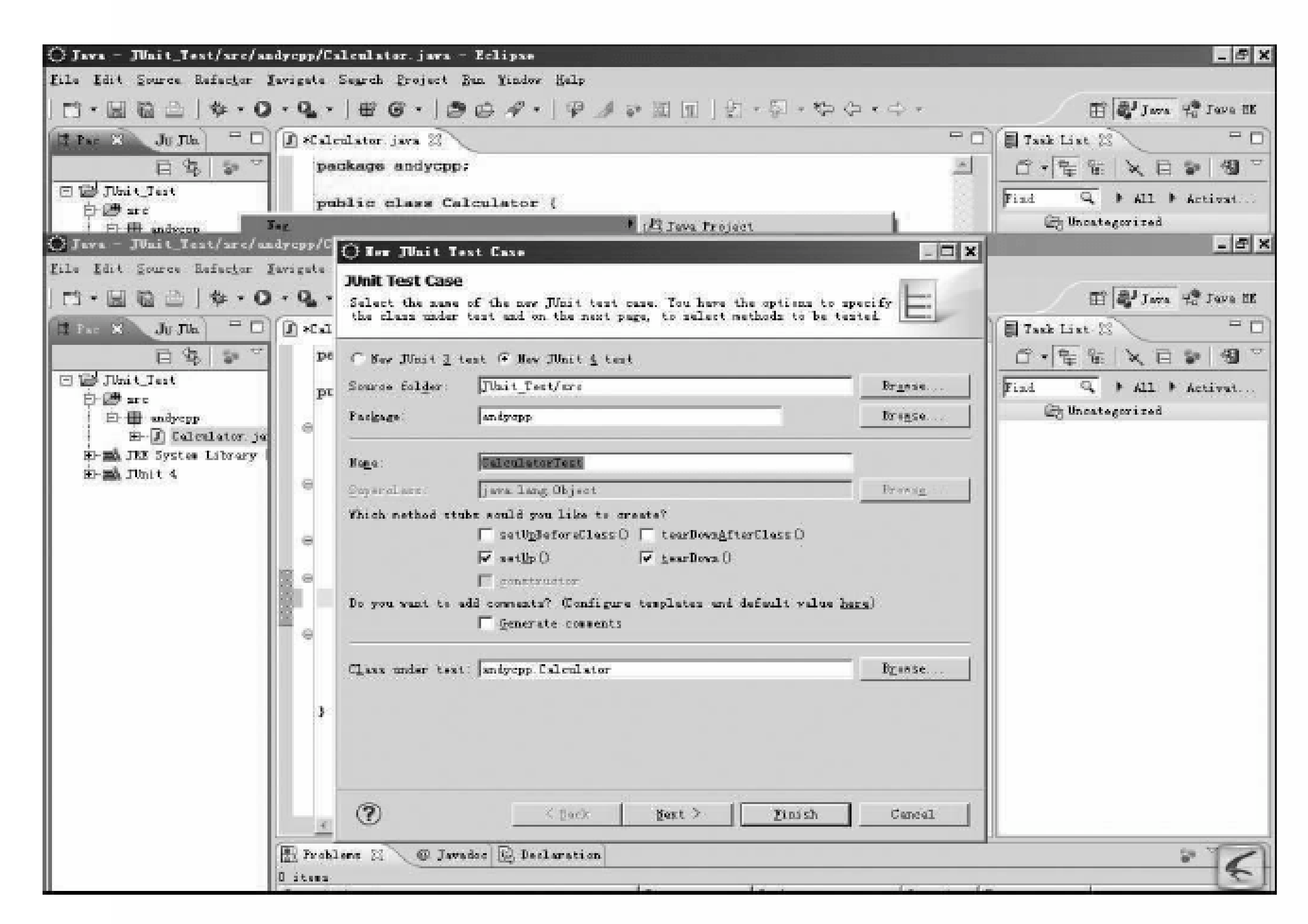


图 14.4 在 eclipse 创建 Calculator 类的测试用例截图 2

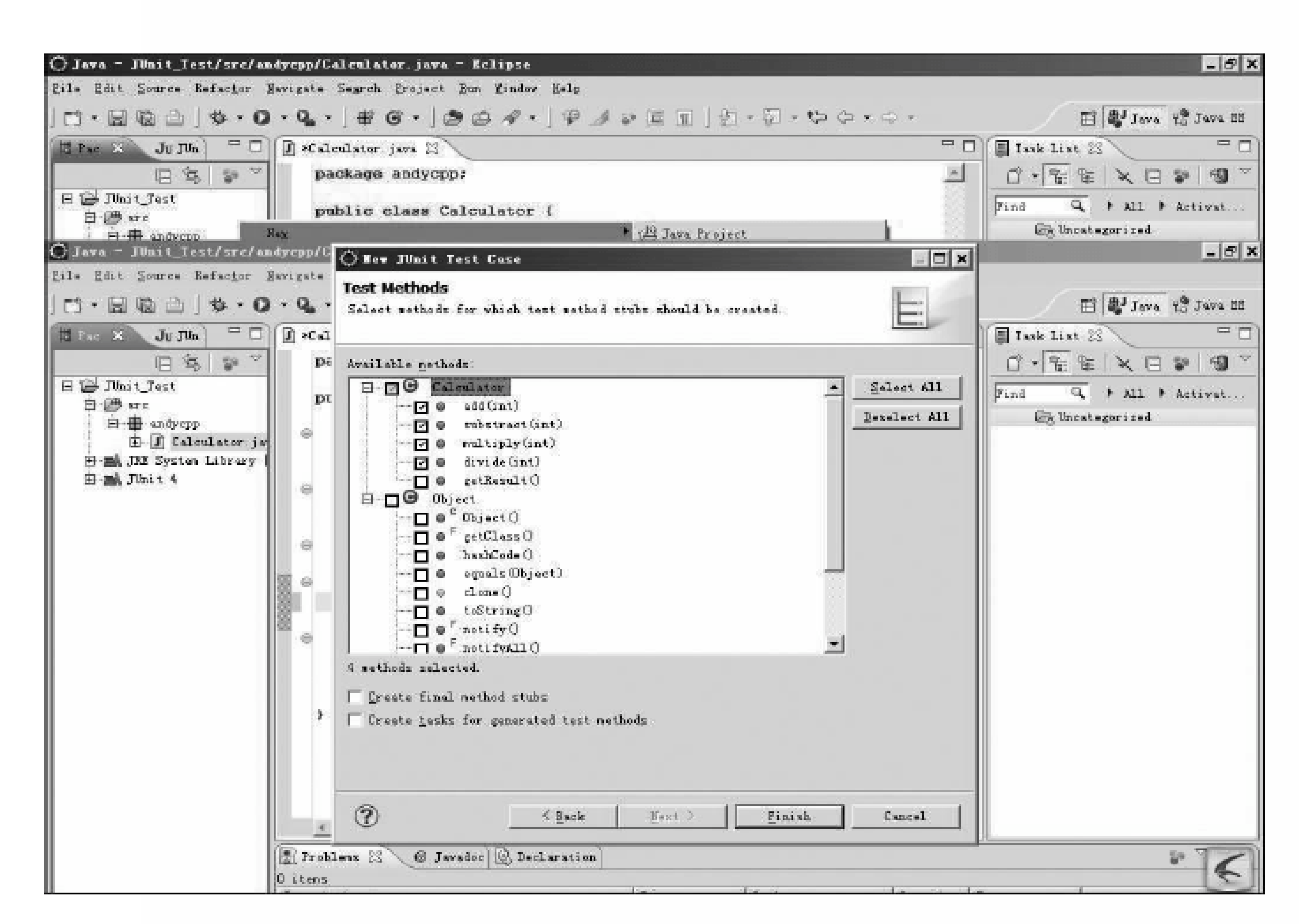


图 14.5 在 eclipse 创建 Calculator 类的测试用例截图 3

CalculatorTest 进行修改，完整代码如下：

```
package andycpp;

import static org.junit.Assert.* ;
import org.junit.Before;
import org.junit.Ignore;
import org.junit.Test;

public class CalculatorTest {
```

```
    private static Calculator calculator=new Calculator();

    @Test
    public void testAdd() {
        calculator.add(2);
        calculator.add(3);
        assertEquals(5, calculator.getResult());
    }

    @Test
    public void testSubstract() {
        calculator.add(10);
        calculator.substract(3);
        assertEquals(7, calculator.getResult());
    }

    @Ignore("Multiply() Not yet implemented")
    @Test
    public void testMultiply() {
    }

    @Test
    public void testDivide() {
        calculator.add(6);
        calculator.divide(2);
        assertEquals(3, calculator.getResult());
    }
}
```

④ 运行测试代码。在 CalculatorTest 类上右击，选择 Run As a JUnit Test，如图 14.6 所示。

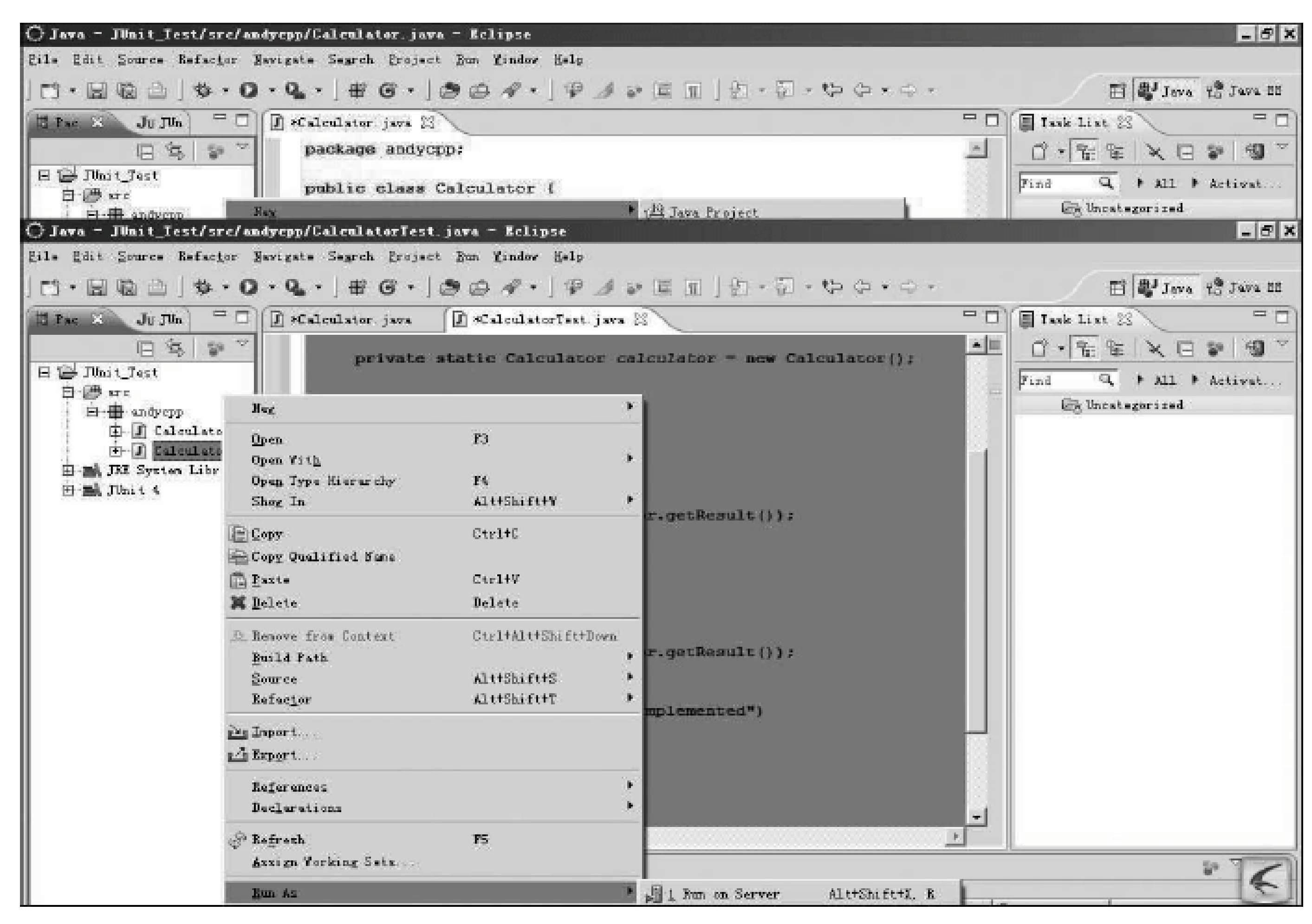


图 14.6　运行测试用例截图 1

运行结果如图 14.7 所示。

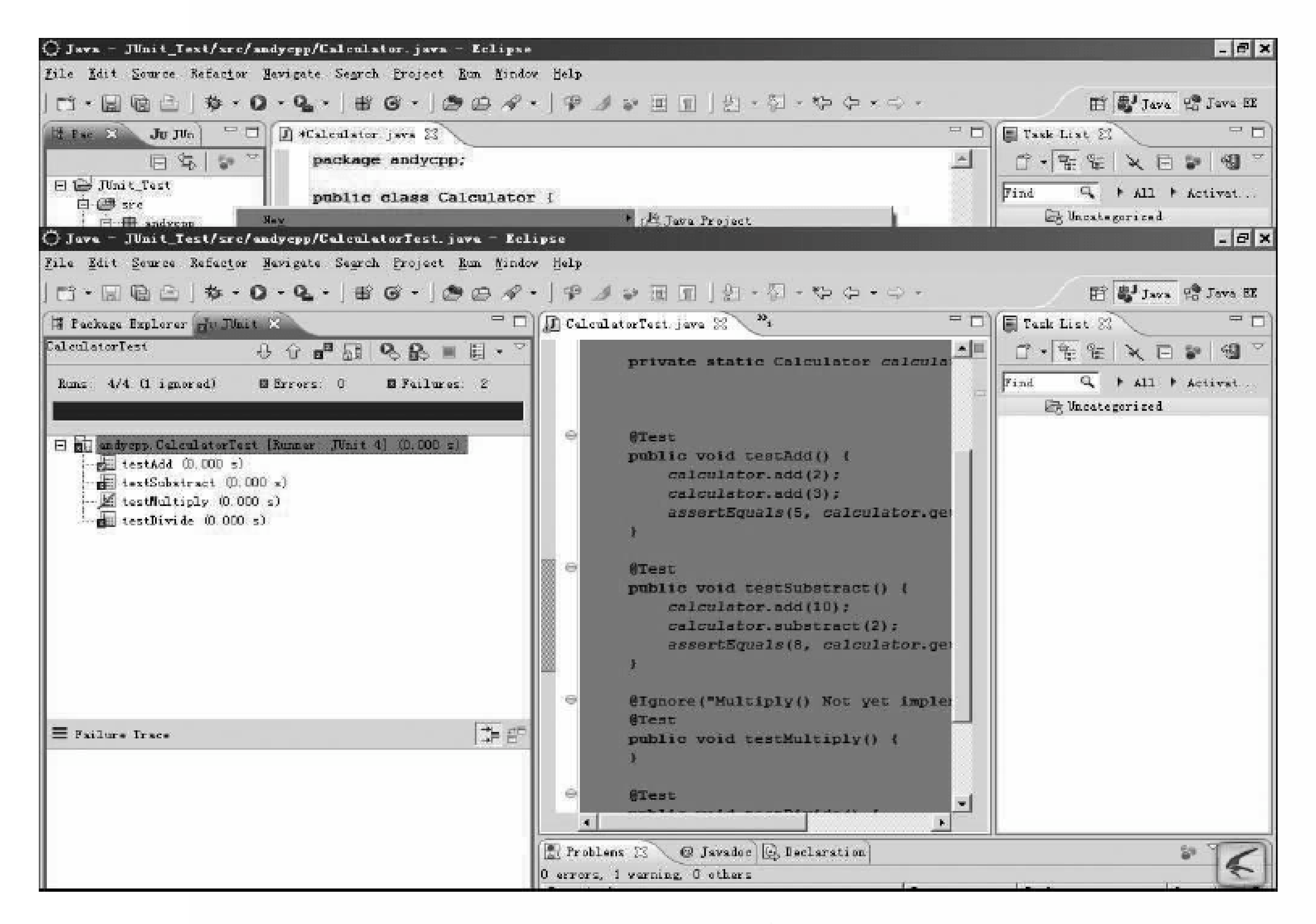


图 14.7　运行测试用例截图 2

图 14.7 中进度条中红颜色表示发现错误，具体的测试结果为“共进行了 4 个测试，其中 1 个测试被忽略，2 个测试失败”。



功能测试工具

功能测试工具利用脚本的录制/回放原理，模拟用户的操作，判断将被测系统的输出结果。功能测试工具有：Rational 公司的 TeamTest 、Robot；Compuware 公司的 QACenter 等。本章介绍 MI 公司的功能测试工具 WinRunner 和 Quick Test Professional。

15.1 WinRunner

15.1.1 WinRunner 测试模式

WinRunner 通过设置检查点、记录手工操作的各个对象为脚本，通过重放检查其在相同的环境中有无异常的现象或与实际结果不符的地方，是否有新错误被引入等。当用户在软件操作中单击图形用户界面上的对象时，WinRunner 会生成类 C 的测试脚本。

WinRunner 包括以下两种录制测试的模式。

1. 环境判断模式

根据用户选取界面上的对象（如窗体、清单、按钮等）的操作动作录制下来，并忽略这些对象在屏幕上的物理位置。每一次对被测软件进行操作，测试脚本中的脚本语言会描述选取的对象和操作动作。

进行录制时，WinRunner 描述用户选取的每个对象写入图形用户界面映射中。由于图形用户界面映射和测试脚本被分开保存，当软件用户界面发生变化时，只需更新图形用户界面映射。执行测试只需要回放测试脚本。WinRunner 模拟一个用户使用鼠标选取对象、用键盘输入数据。WinRunner 从图形用户界面映射中读取对象描述，并在被测软件中查找符合这些描述的对象。

2. 模拟模式

模拟模式记录鼠标点击、键盘输入和鼠标精确的运动轨迹。执行测试时，WinRunner 让鼠标根据轨迹运动。

15.1.2 WinRunner 测试流程

WinRunner 的测试过程分为创建 GUI 映射、创建测试 、调试测试 、执行测试 、查看测试结果和报告发现的错误等六个步骤，分别如下：

(1) 创建 GUI map

使用快速测试脚本向导回顾软件用户界面，并系统地把每个 GUI 对象的描述添加到 GUI 映射中。也可以在录制测试的时候，通过单击对象把对单个对象的描述添加到 GUI map 中。

(2) 创建测试

用户可以通过录制、编程等方式创建测试脚本，可以插入检查点来检查 GUI 对象。

(3) 调试测试

用户可在调试模式下运行脚本，设置中断点，监测变量。

(4) 执行测试

检验模式下测试被测软件。WinRunner 在脚本运行中遇到检查点后，把当前数据和前期捕捉的期望值进行比较。

(5) 查看测试结果

每次测试结束，WinRunner 会把结果显示在报告中，报告会详述测试执行过程中发生的所有主要事件，如检查点、错误信息、系统信息或用户信息。如果在检查点有不符合被发现，可以在“测试结果”窗口查看预期结果和实测结果。如果是位图不符合，也可以查看用于显示预期值和实测结果之间差异的位图。

(6) 报告发现的错误

如果由于测试中发现错误而造成测试运行失败，用户可以从“测试结果”窗口报告有关错误的信息。这些信息通过 EMAIL 发送给测试经理，用来跟踪这个错误直到被修复。

下面，通过 WinRunner 自带的飞机订票的例子，讲述 WinRunner 的操作。

(1) 创建 GUI map

当创建测试时，用户需要确定 GUI map 的工作模式。GUI map 有两种工作模式：Global GUI Map File 和 GUI Map File per Test。

GUI Map File per Test 模式每次新建测试就自动新建一个 GUI map file。更改 GUI map 的工作模式：在 WinRuner 主菜单栏 Tools→General Options→GUI Files 栏选择 GUI Map File per Test 模式，如图 15.1 所示。

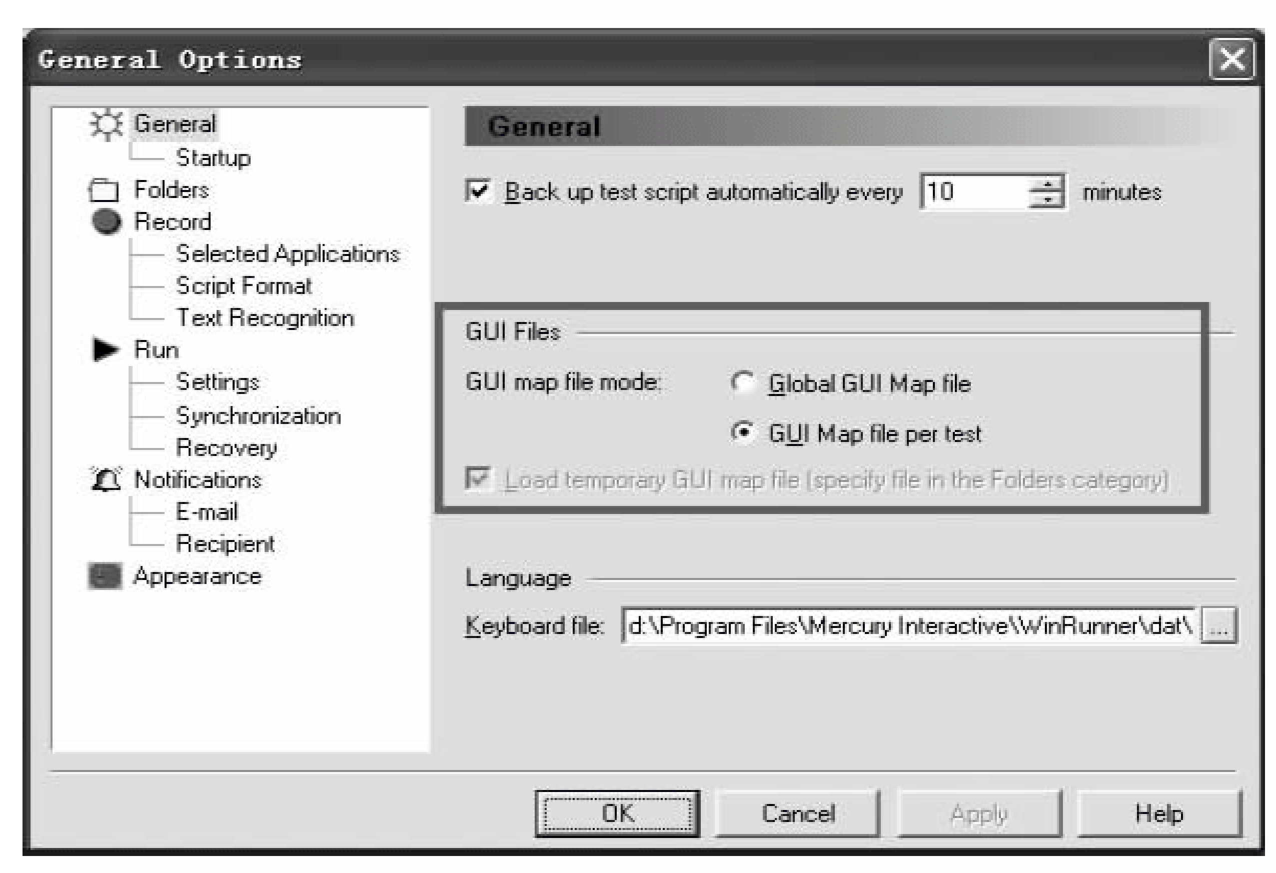


图 15.1 WinRunner 更改 GUI map 工作模式界面

（2）创建测试

① 运行 WinRunner，新建一个测试项目：在 WinRuner 主菜单栏选择 File→New。

② 运行 Flight 4A 程序，进入 Login 窗口：单击“开始”按钮，选择“开始”→“程序”→WinRunner→Sample Applications，单击 Flight 4A 快捷方式，显示出示例程序的登录界面，如图 15.2 所示。

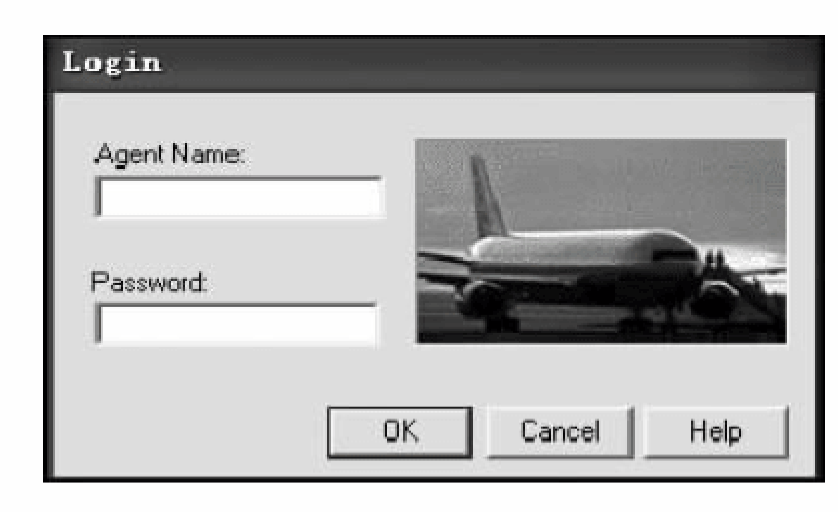


图 15.2 示例程序 Flight 4A 登录界面

③ 开始录制：在 WinRunner 工具栏单击 Record 按钮，WinRunner 进入录制状态。

④ 执行登录程序：在 Flight 4A 示例程序登录界面，单击 Agent Name 输入框，输入 admin，单击 Password 输入框，输入 mercury，单击 OK 按钮，程序登录成功，进入主界面。

⑤ 停止录制：在 WinRunner 主菜单栏点击“Stop”按钮，可以看到 WinRunner 中记录了脚本如下：

```
1 #Login
2 set_window ("Login", 4);
3 edit_set ("Agent Name:", "admin");
4 obj_type ("Agent Name:","<kTab>");
5 password_edit_set("Password:", "kzptnyoslzjsaz");
6 button_press ("OK");
```

代码分析如下。

第一行：注释。

第二行：获取登录 Login 对话框。

第三行：在“Agent Name:”编辑框中输入 admin。

第四行：用户按了 Tab 键。

第五行：在“ Password:”编辑框中输入密码 mercury，脚本显示为加密后的文本。

第六行：用户按了 OK 按钮。

⑥ 保存脚本：保存脚本为 C:\TestProject\WR_Lesson_Record_and_Run。GUI Map 可以自动保存在用户目录下，也可选择手动保存，如图 15.3 所示。

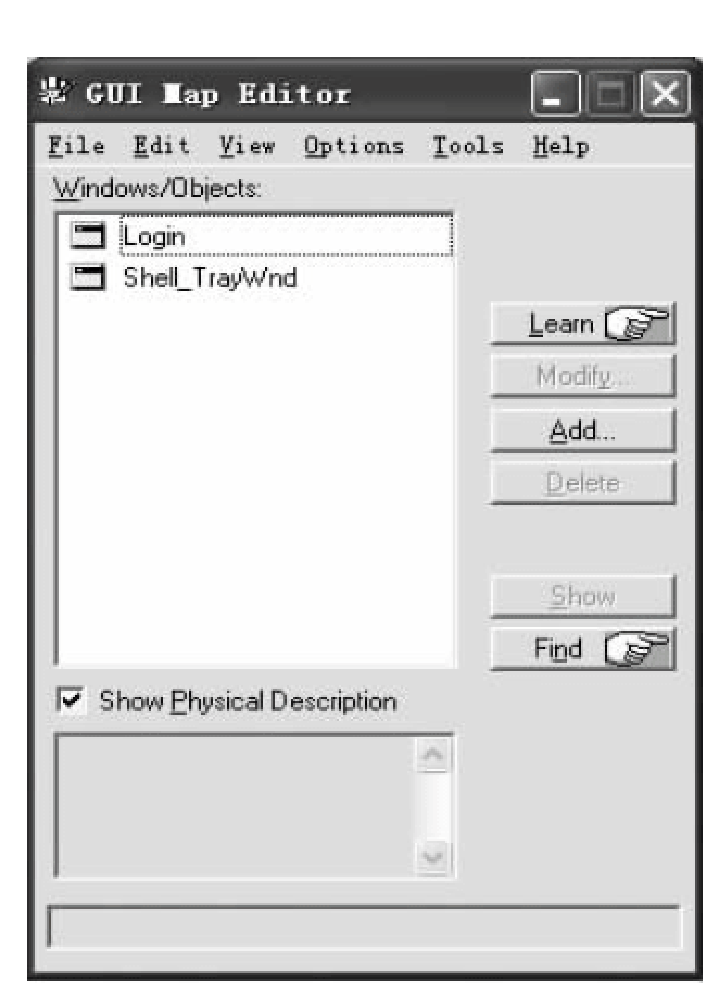


图 15.3 WinRunner GUI Map 文件保存界面

（3）修改测试脚本

录制一个脚本的时候，每次点击 GUI 对象或用键盘输入时，在 WinRunner 的测试脚本中产生了一个 TSL 的声明。测试人员可以通过函数发生器为脚本定义函数或则通过其选择相应的函数。

```
#Login
set_window ("Login", 4);
```

```
edit_set ("Agent Name:", "admin");
obj_type ("Agent Name:","<kTab>");
wait(2);                                              '等待 2 秒
password_edit_set("Password:", "kzptnyoslzjsaz");
button_press ("OK");
```

(4) 执行测试

① 关闭 Flight 4A 程序登录后进入的 Flight Reservation 对话框。

② 重新运行 Flight 4A 程序,等待 Login 对话框的出现。

③ 回放刚才录制的脚本:在 WinRunner 主菜单栏单击 Run from Top。可以看到刚才对 Login 窗口所做的操作和输入均被重现了一遍。由于加入了 wait(2)函数,所以输入用户名后,等待 2 秒后才输入密码。

执行所记录的测试脚本和分析运行结果的时候,WinRunner 提供三种运行模式,可以从工具栏中选择运行模式:

- 运行测试脚本,检测应用程序的运行情况,察看各检查点内容是否和预期一致,并保存测试结果,使用"校验"模式。该模式为默认运行模式。
- 检验测试脚本是否存在语法错误,使用"调试"模式。
- 更新已创建的用户界面检查点和位图检查点的预期值,使用"校正"模式。

(5) 查看测试结果

一次测试运行结束以后,在 WinRunner 的测试结果窗口, WinRunner 将不同运行结果标以不同颜色(绿色的显示正确结束,红色的显示失败),如图 15.4 所示。

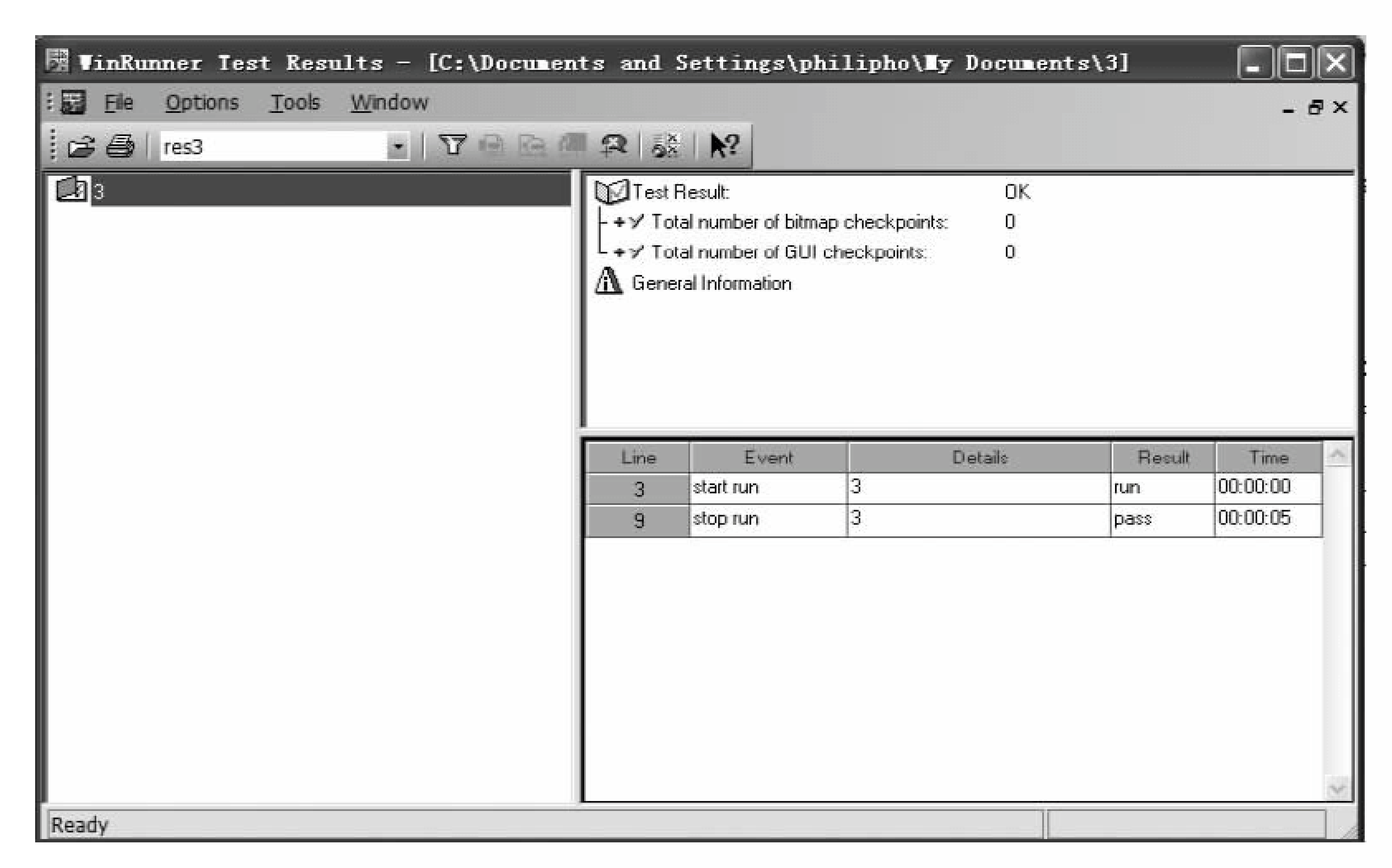


图 15.4 WinRunner 测试结果界面

15.1.3 WinRunner 测试举例

WinRunner 可以检查对象的单个属性是否与预期的相一致,下面用 WinRunner 自

带的 Flight 4A 为预期目标，Flight 4B 为有错误的对比程序进行讲解，详细步骤如下所示。

（1）创建 GUI map

使用 GUI Map File per Test 模式。

（2）创建测试

① 运行 WinRunner，新建一个测试项目。

② 运行 Flight 4A 程序，进入示例程序主界面如图 15.5 所示。



图 15.5　示例程序 Flight 4A 主界面

③ 开始录制：在 WinRunner 主菜单栏单击 Record 按钮，WinRunner 进入录制状态。

④ 进入 Flight 4A Open Order 对话框：在 Flight 4A 示例程序对话框，选择 File→Open Order，如图 15.6 所示。

⑤ 选择 Order No，此时 Customer Name 复选框应置灰，如图 15.7 所示。

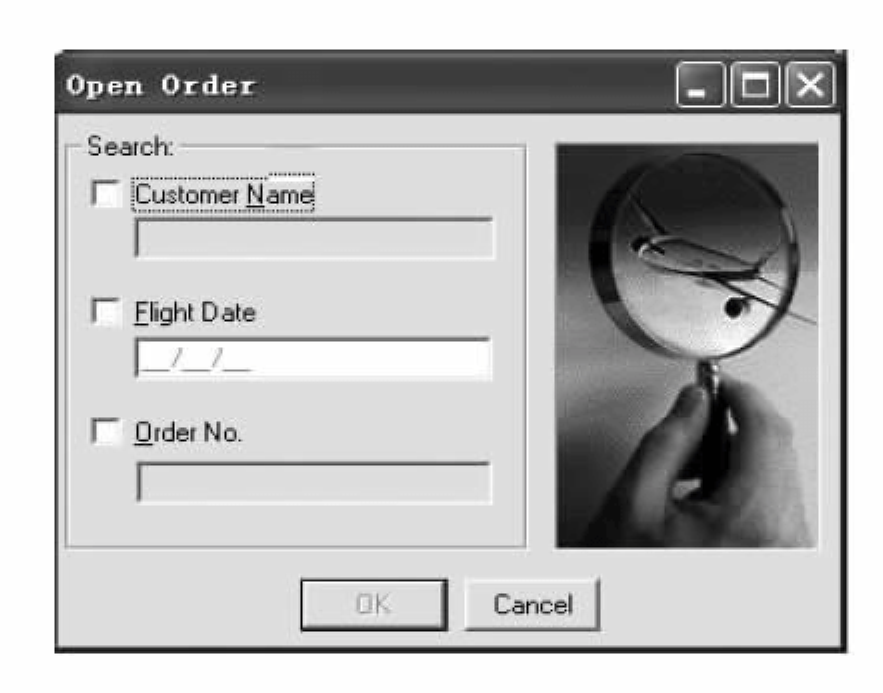


图 15.6　Open Order 对话框

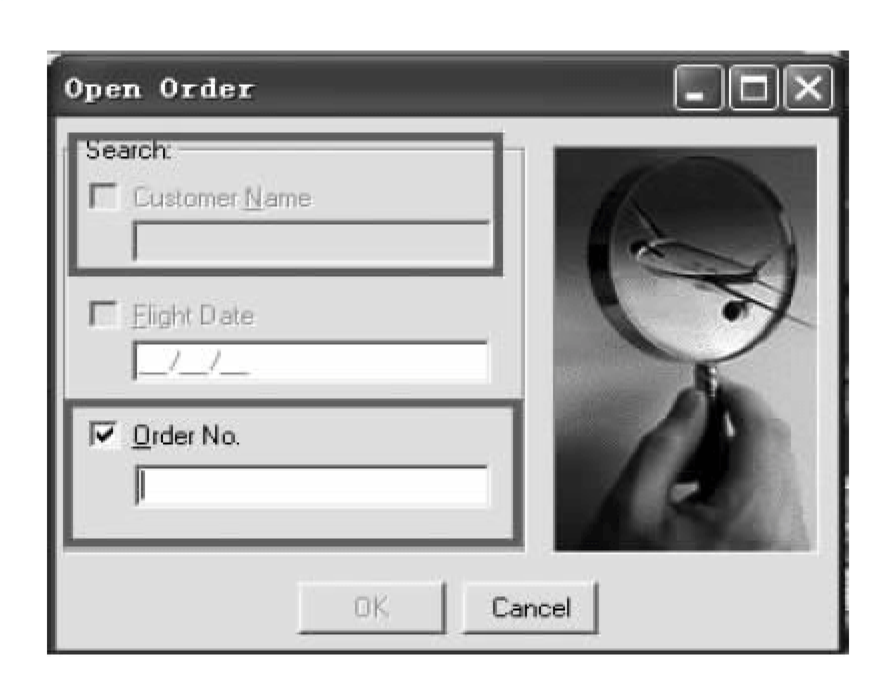


图 15.7　示例程序 Flight 4A 点击 Order No 界面

⑥ 检查 Customer Name 复选框 enable 属性：选择 WinRunner 菜单 Insert→GUI Checkpoint→For Single Property，进入选取目标状态，如图 15.8 所示。

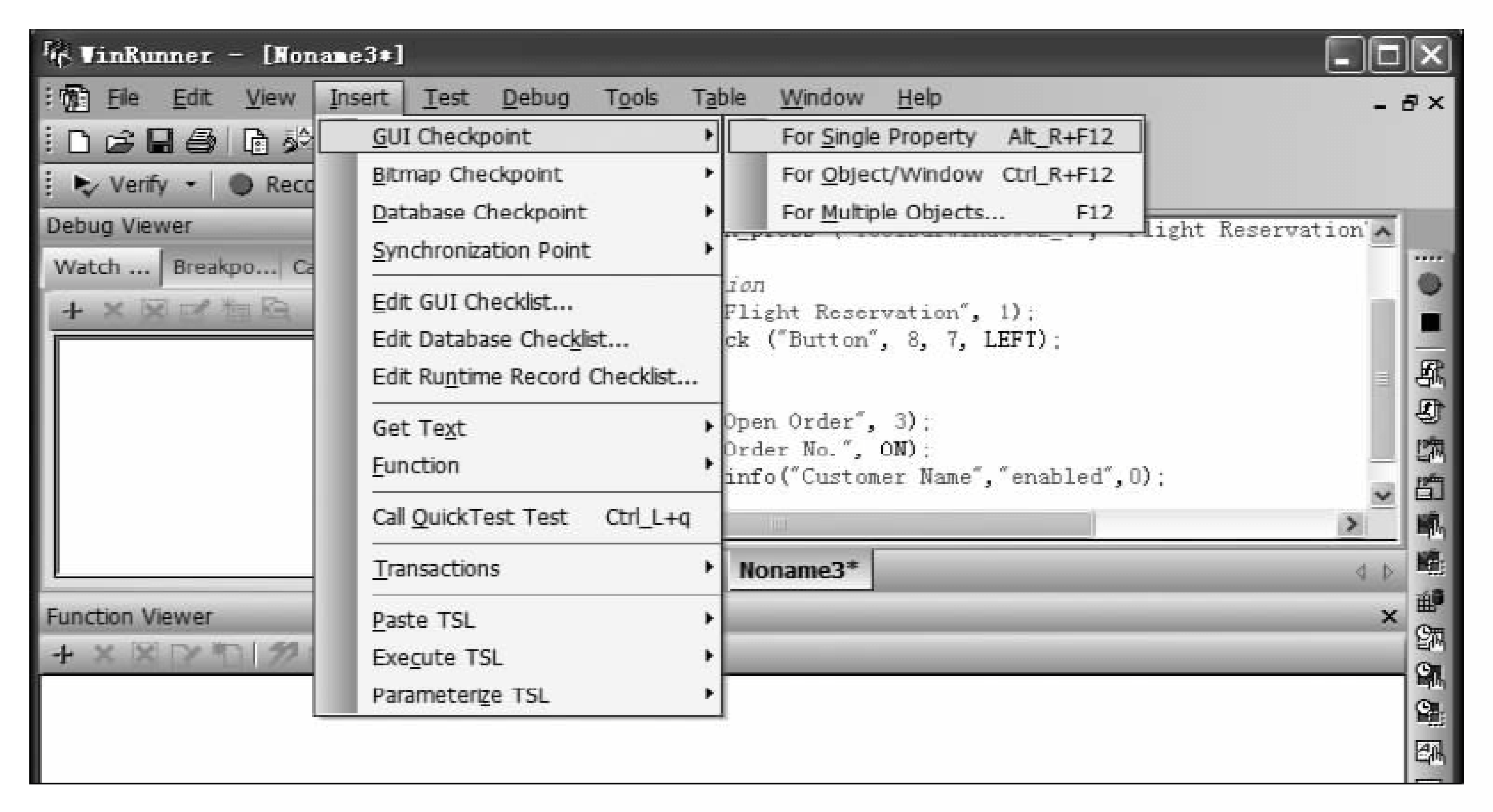


图 15.8　WinRunner 选取检查属性设置界面

⑦ 移动形光标，在 Flight 4A 示例程序 Customer Name 复选框上单击左键，显示检查属性窗口，如图 15.9 所示。

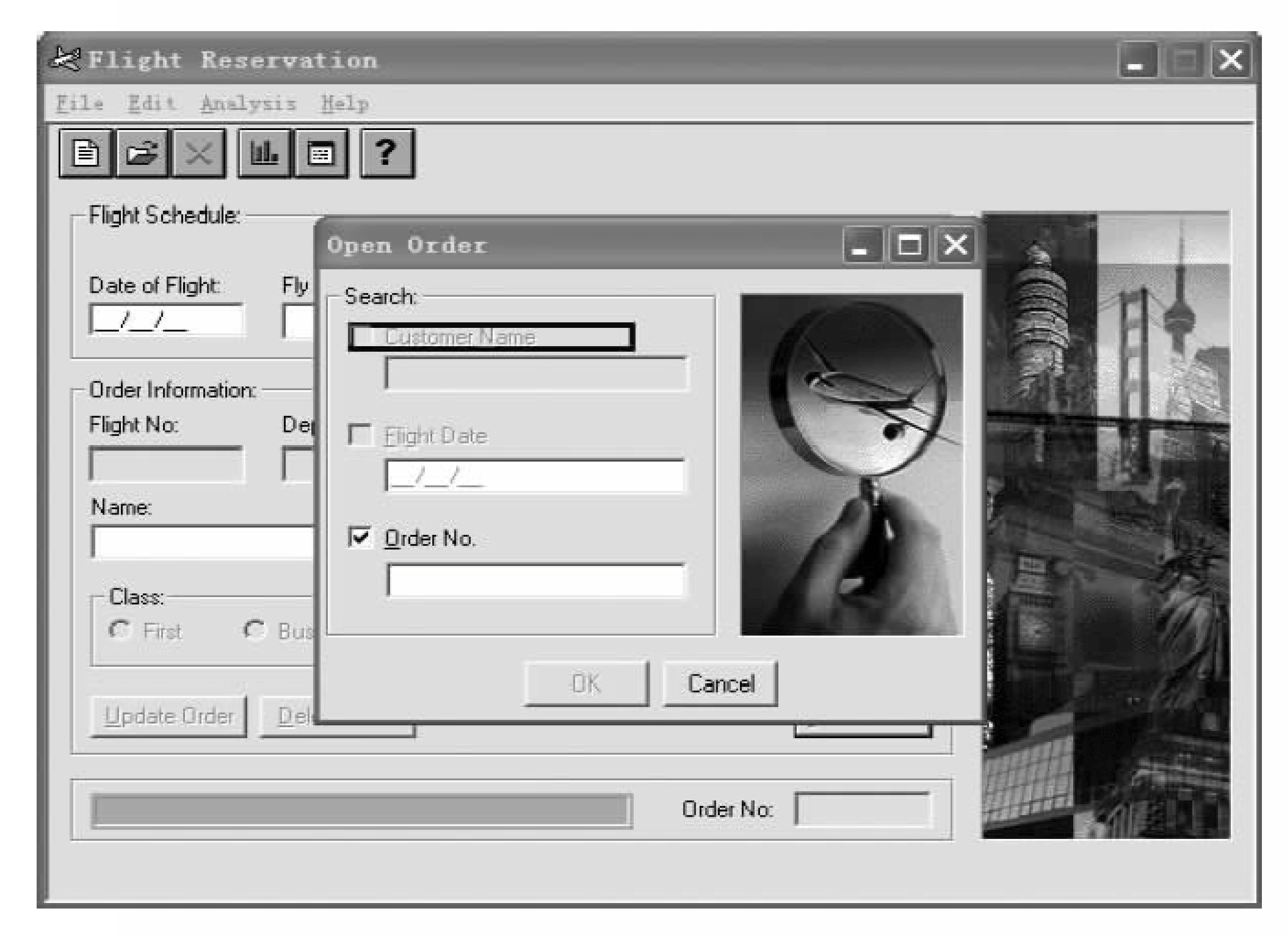


图 15.9　WinRunner 检查属性选取界面

⑧ 可以看到要检查的属性为 enabled，确认预期值为 0（相当于 False），如图 15.10 所示。

⑨ 在 WinRunner 检查属性窗口单击 Paste 按钮，可以看到脚本中增加了一行语句：

```
button_check_info("Customer Name","enabled",0);
```

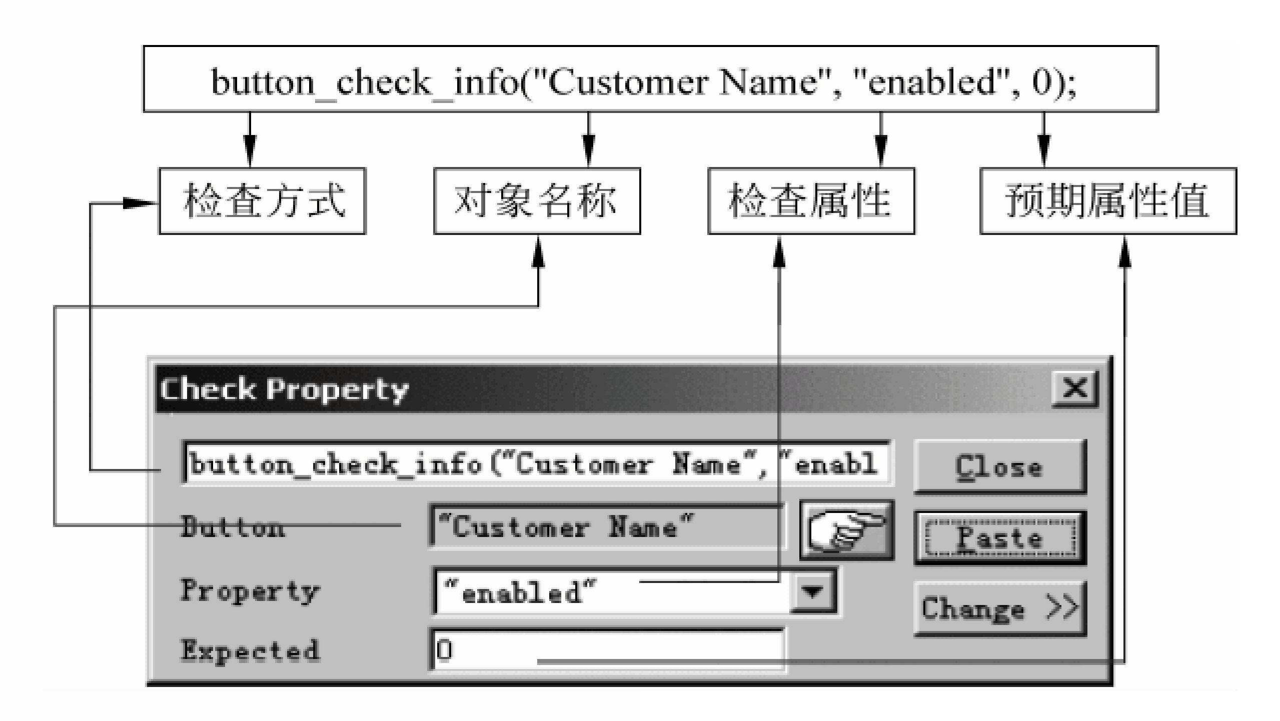


图 15.10　WinRunner 检查属性设置界面

⑩ 停止录制：在 Flight 4A 程序单击 Cancel 按钮返回至主界面；在 WinRunner 主菜单栏单击 Stop 按钮，可以看到 WinRunner 中记录了脚本如下：

```
1 #Flight Reservation
2 set_window ("Flight Reservation", 2);
3 menu_select_item ("File;Open Order...");
4
5 #Open Order
6 set_window ("Open Order", 2);
7 button_set ("Order No.", ON);
8 button_check_info("Customer Name","enabled",1);
```

代码结束如下：

第一行：注释。

第二行：获取登录 Flight Reservation 示例程序主界面。

第三行：选择 File→Open Order 子菜单。

第六行：用户获取 Open Order 界面。

第七行：用户选择 Order No。

第八行：检查属性的设置。

⑪ 保存脚本：保存脚本为"C:\TestProject\WR_Lesson_Cherck_Single"。

(3) 修改测试脚本

根据需求用 TSL 脚本语言手工编程方式检查属性，修改测试脚本达到测试的要求，在本例中不做修改。

(4) 执行测试

① 确认 Flight 4A 程序登录后进入的 Flight Reservation 窗口。

② 回放刚才录制的脚本，在主菜单栏单击 Run from Top，查看所做的操作均被重现了一遍。

(5) 查看测试结果

当一次测试运行结束以后，在 WinRunner 的测试结果窗口可以看到没有错误，全部

通过，如图 15.11 所示。

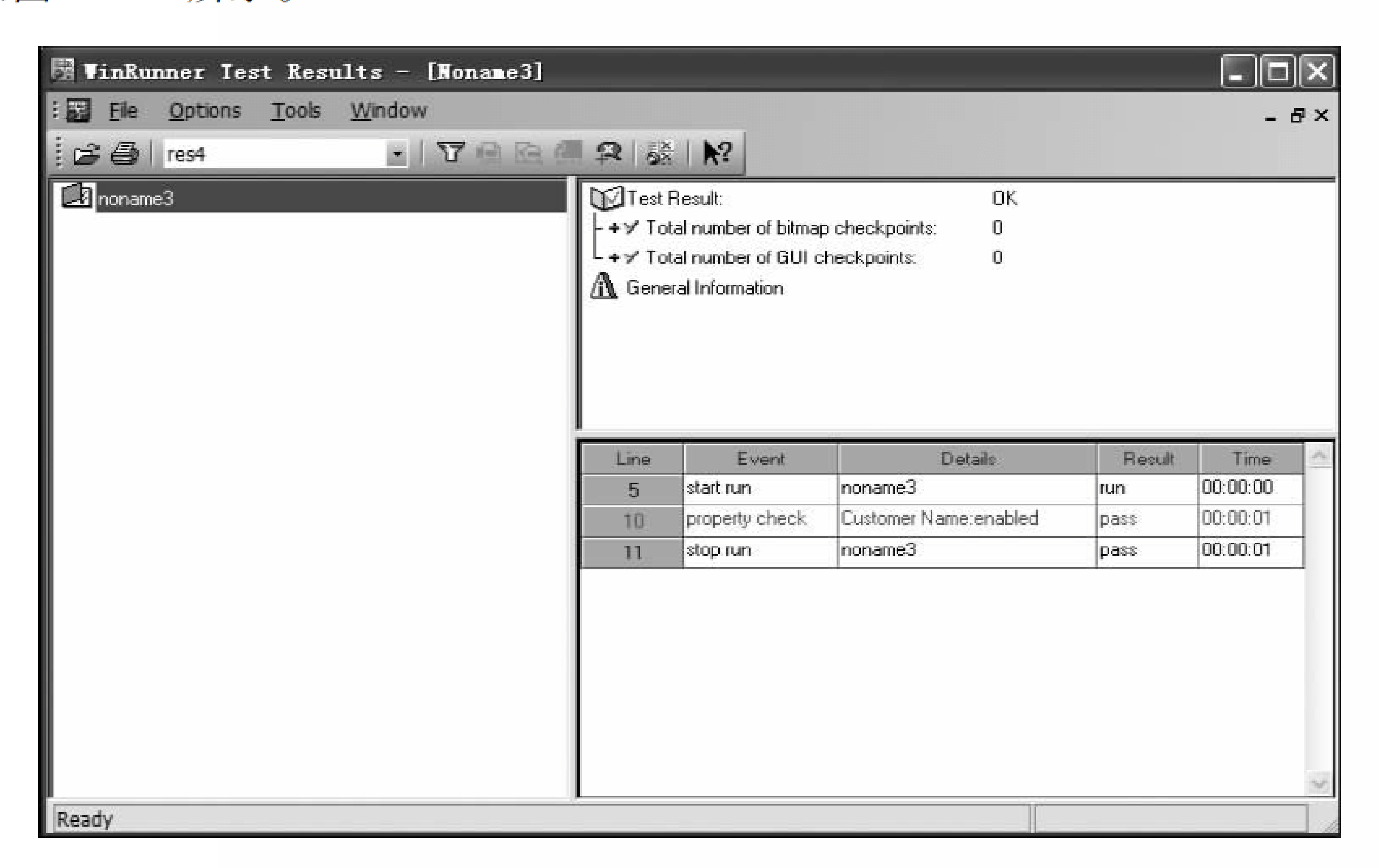


图 15.11　WinRunner 测试结果界面

(6) 执行测试

① 确认退出 Flight 4A 程序。

② 确认 Flight 4B 程序登录后进入的 Flight Reservation 窗口。

③ 回放刚才录制的脚本，在 WinRunner 主菜单栏单击 Run from Top，可以看到执行过程中出现了错误。单击 Order No 按钮，Customer Name 复选框没有置灰，如图 15.12 所示。

(7) 查看测试结果

在 WinRunner 的测试结果窗口可以看到检查点的记录为红色，结果为 fail，双击后可以看到原因为与预期的结果不一致，如图 15.13 所示。



图 15.12　示例程序 Flight 4A 单击 Order No 界面

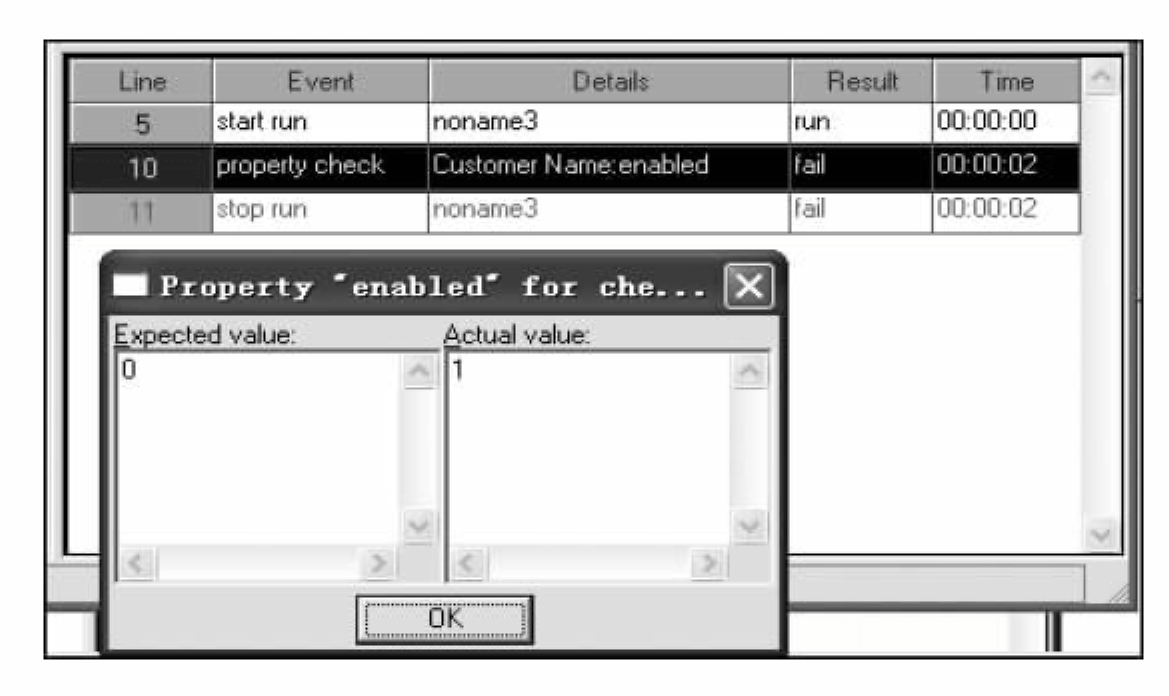


图 15.13　WinRunner 测试结果界面

15.2 QuickTest Professional 简介

QuickTest Professional 缩写为 QTP，由 Mercury 公司开发，用于功能测试和回归测试自动化。QTP 采用关键字驱动测试解决方案，大大简化了测试的创建和维护。

15.2.1 QuickTest Professional 测试过程

QuickTest Professional 的测试过程分为准备录制、录制测试脚本、加强测试脚本、调试测试、执行测试、查看测试结果和报告发现的错误。

(1) 准备录制

在录制测试前，检查应用程序和 QTP 已按测试要求设置，并确保应用程序显示要录制的元素，例如，工具栏或特殊窗口窗格；还要确保应用程序选项已按测试目标设置。

(2) 录制测试脚本

操作应用程序或浏览网站时，QTP 会以表格的方式显示录制的操作步骤。每一个操作步骤都是使用者在录制时的操作，如在网站上点击了链接，或则在文本框中输入的信息。

(3) 加强测试脚本

在测试脚本中加入检查点，检查网页的链接、对象属性、或者字符串，以验证应用程序的功能是否正确。将录制的固定值以参数取代，使用多组的数据测试程序。使用逻辑或者条件判断式，可以进行更复杂的测试通过在测试中插入检查点可以搜索页面、对象或文本字符串中的特定值，这有助于确定应用程序或网站是否正常运行。

(4) 调试测试

修改过测试脚本后，需要对测试脚本作调试，以确保测试脚本能正常执行。

(5) 执行测试

通过执行测试脚本，QTP 会在新本的网站或者应用程序上执行测试，检查应用程序的功能是否正确。

(6) 查看测试结果

分析测试结果，找出问题所在。

(7) 报告发现的错误

与 TestDirector 集成，将发现问题上传 TestDirector 数据库中。

15.2.2 使用 Mercury Tours 范例网站

用户安装 QTP 后，会自动安装单机版的 Flight Reservation(航班预订)软件，作为用户测试应用程序的范例软件。并提供了 Mercury Tours 示范网站供用户学习 WEB 测试，Mercury Tours 是一个提供机票预订服务的网站，

下面，将 Mercury Tours 示范网站作为演示 QTP 各个功能的例子程序，步骤如下所示。

(1) 优化测试的浏览器设置

使用 Internet Explorer 作为浏览器，则应该清除用户名和密码的“自动完成”选项。

这样将确保在创建测试时，可以精确录制所有的操作。

（2）启动 Mercury Tours 应用程序

在 Web 浏览器中，键入以下 URL：http://newtours.demoaut.com，将打开 Mercury Tours 主页，如图 15.14 所示。

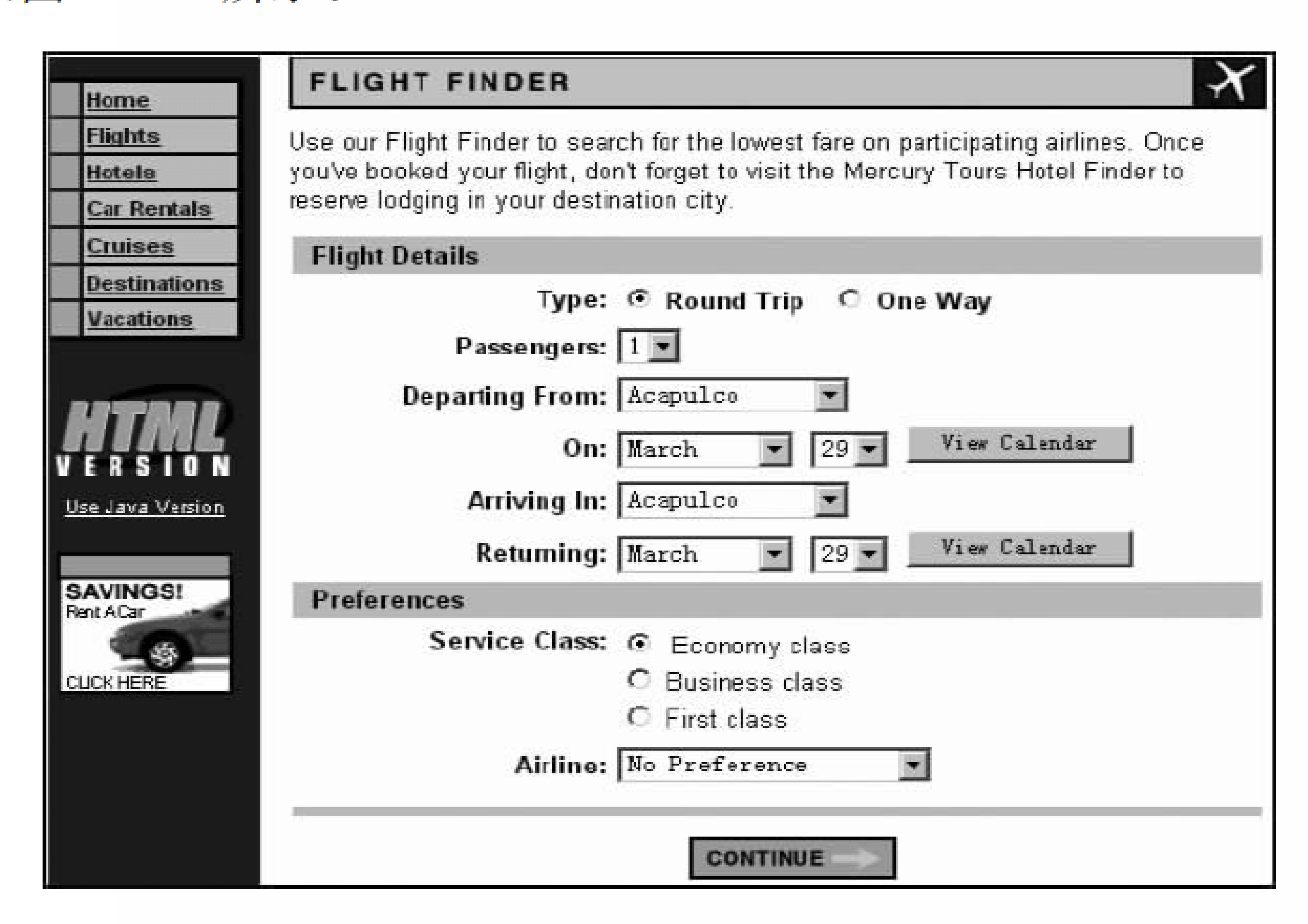


图 15.14　Mercury Tours 范例网站界面

（3）在 Mercury Tours 中注册

要登录并使用 Mercury Tours 应用程序，必须成为注册用户。在主页上，单击 REGISTER 导航按钮。打开 Register 页。

（4）浏览 Mercury Tours 站点

从 Flight Finder 页开始，按照屏幕上的说明获得航班信息并预定航班。

（5）结束 Mercury Tours 会话

在浏览 Mercury Tours 应用程序完成后，单击 Flight Confirmation 页上的 LOG OUT 按钮，或单击任何应用程序页顶部的 SIGN-OFF 链接。

15.2.3　QTP 测试范例

下面，通过 QTP 自带的飞机订票网站的例子，来讲述 QTP 的使用。具体步骤如下所示。

（1）录制测试脚本

应用 QTP 录制一个测试脚本，在 Mercury Tours 范例网站上预定一张从纽约（New York）到旧金山（San Francisco）的机票。

① 启动 QTP：请选择“开始”→“程序”→QuickTest Professional→QuickTest Professional。在“加载项管理器”中，确认 Web 加载项处于选定状态，并清除所有其他加载项。单击“确定”按钮，关闭“加载项管理器”，并打开 QTP。

② 新建一个测试项目：在主菜单栏选择 File→New 按钮。

③ 开始录制测试脚本：在工具栏上单击 Record 按钮，打开 Record and Run Settings 对话框，如图 15.15 所示。

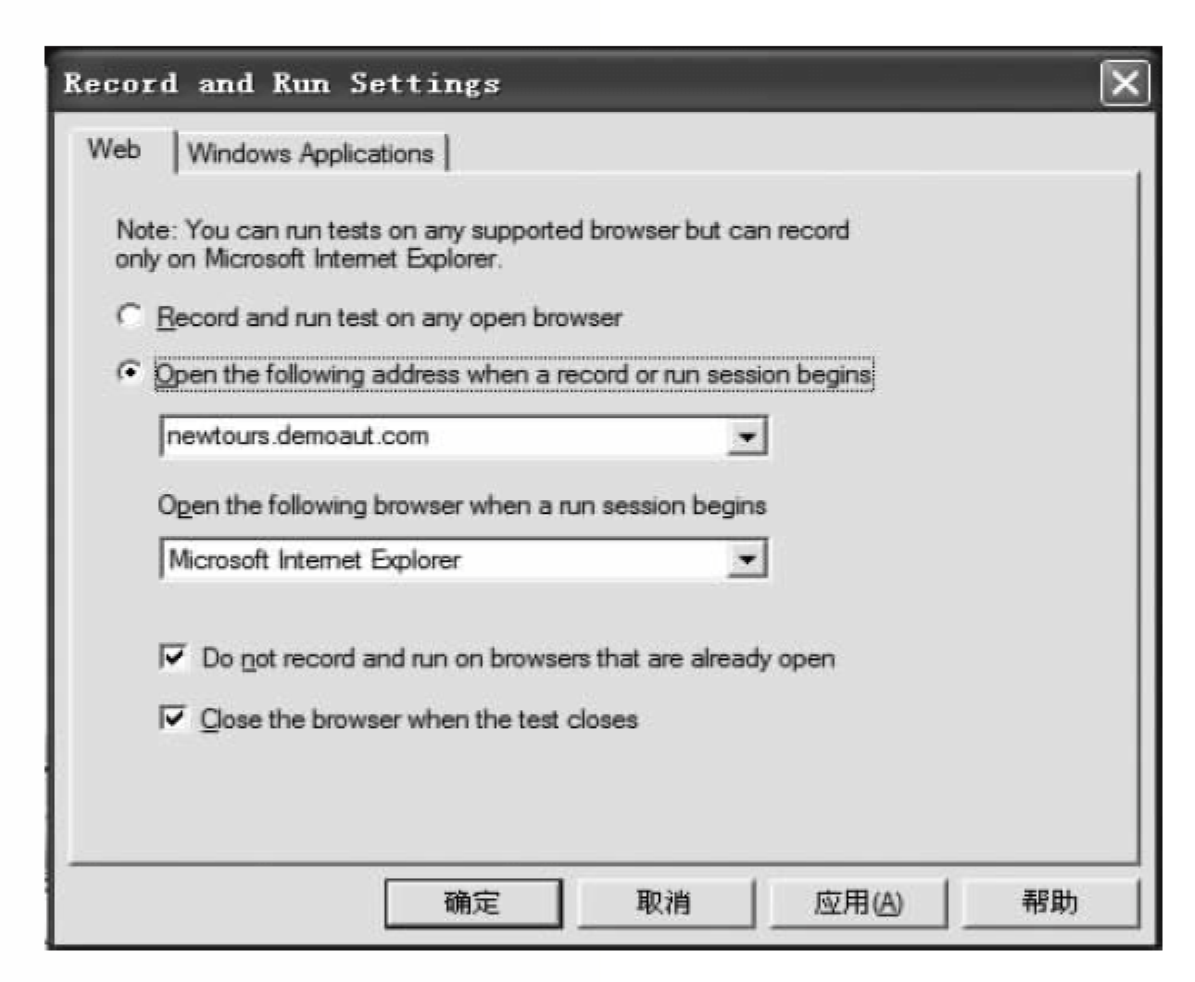


图 15.15 QTP 录制脚本设置界面

④ 切换到 Windows Application 选项卡，如图 15.16 所示。

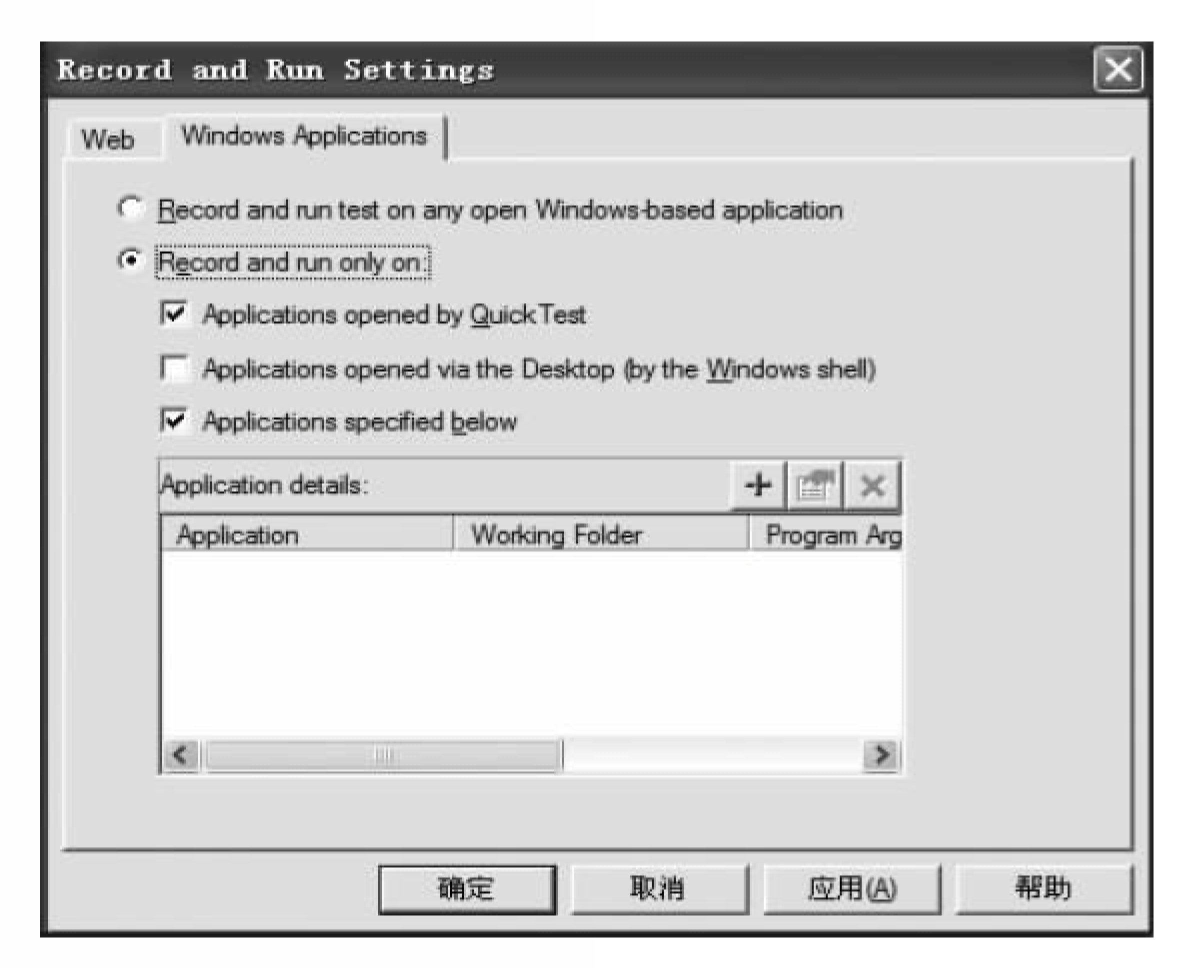


图 15.16 QTP 录制脚本设置界面

⑤ 选择第二个单选按钮。因为只是对 Mercury Tours 范例网站进行操作，不涉及 Windows 程序，所以保持列表为空。

⑥ 单击“确定”按钮，开始录制。

⑦ 登录 Mercury Tours 网站在用户名和密码输入注册时使用的账号和密码，单击 Sign-in，进入 Flight Finder 网页。

⑧ 输入订票数据输入以下订票数据：

```
Departing From: New York
On: May 14
Arriving In: San Francisco
Returning: May 28
Service Class: Business class
```

其他字段保留默认。

单击 CONTINUE 按钮，打开 Select Flight 页面。

⑨ 选择飞机航班：单击 CONTINUE 按钮，打开 Book a Flight 页面。

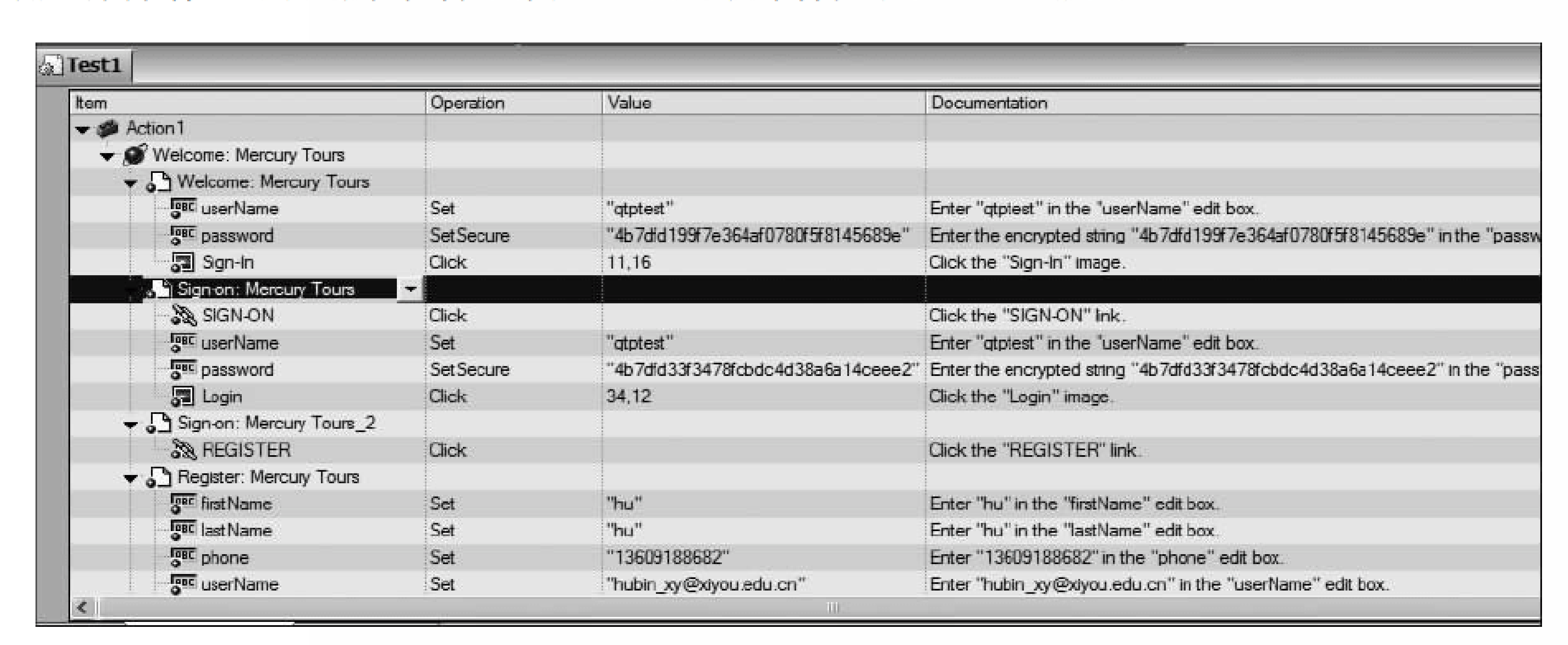
⑩ 查看订票数据，并选择 BACK TO HOME，回到 Mercury Tours 网站首页。

⑪ 停止录制：在 QTP 工具列上单击 Stop 按钮，停止录制。到这里已经完成了预定从“纽约-旧金山”机票的动作，并且 QTP 已经录制了从按下 Record 按钮后到 Stop 按钮之间的所有操作。

⑫ 保存脚本：单击工具栏上的 Save 按钮，开启 Save 对话框。选择路径，填写文件名，取名为 Flight。单击“保存”按钮进行保存。

（2）分析测试脚本

在录制过程中，QTP 会在测试脚本管理窗口中产生对每一个操作的相应记录。并在 Keyword View 中显示所录制的测试脚本。当录制结束后，QTP 也就记录下了测试过程中的所有操作。测试脚本管理窗口显示的内容如图 15.17 所示。



Test1

Item	Operation	Value	Documentation
Action1			
Welcome: Mercury Tours			
Welcome: Mercury Tours			
userName	Set	"qtptest"	Enter "qtptest" in the "userName" edit box.
password	SetSecure	"4b7dfd199f7e364af0780f5f8145689e"	Enter the encrypted string "4b7dfd199f7e364af0780f5f8145689e" in the "passw
Sign-In	Click	11,16	Click the "Sign-In" image.
Sign-on: Mercury Tours			
SIGN-ON	Click		Click the "SIGN-ON" link.
userName	Set	"qtptest"	Enter "qtptest" in the "userName" edit box.
password	SetSecure	"4b7dfd33f3478fcbdc4d38a6a14ceee2"	Enter the encrypted string "4b7dfd33f3478fcbdc4d38a6a14ceee2" in the "pass
Login	Click	34,12	Click the "Login" image.
Sign-on: Mercury Tours_2			
REGISTER	Click		Click the "REGISTER" link.
Register: Mercury Tours			
firstName	Set	"hu"	Enter "hu" in the "firstName" edit box.
lastName	Set	"hu"	Enter "hu" in the "lastName" edit box.
phone	Set	"13609188682"	Enter "13609188682" in the "phone" edit box.
userName	Set	"hubin_xy@xiyou.edu.cn"	Enter "hubin_xy@xiyou.edu.cn" in the "userName" edit box.

图 15.17　QTP 脚本管理窗口

在 Keyword View 中的每一个字段都有其意义：

- Item：以阶层式的图标表示操作步骤所作用的组件。
- Operation：要在这个作用到的组件上执行的动作，如点击、选择等。
- Value：执行动作的参数，例如当鼠标点击一张图片时是用左键还是右键。
- Assignment：使用到的变量。
- Comment：在测试脚本中加入的批注。
- Documentation：自动产生用来描述此操作步骤的英文说明。

脚本中的每一个步骤在 Keyword View 中都会以一列来显示，其中用来表示此组件

类别的图标以及步骤的详细数据。

图 15.18 是针对一些常见的操作步骤作详细说明：

步骤	描述
Action1	Action1 是操作名。
Welcome: Mercury Tours	浏览器调用 Welcome:Mercury Tours 网站。
Welcome: Mercury Tours	Welcome:Mercury Tours 是网页的名称。
userName Set "mercury"	userName 是编辑框的名称。Set 是在编辑框上执行的方法。mercury 是编辑框的值。
password SetSecure "4082820183...	password 是编辑框的名称。SetSecure 是在编辑框上执行的加密方法。4082820183afe512e8bc91c1f7222dbd 是 password 的加密值。
Sign-In Click 2,2	Sign-In 是图像链接的名称。Click 是在图像上执行的方法。2, 2 是图像单击位置的 x 坐标和 y 坐标。

图 15.18　QTP 操作步骤说明

(3) 执行测试

当运行录制好的测试脚本时，QTP 会打开被测试程序，执行在测试中录制的每一个操作。测试运行结束后，QTP 显示本次运行的结果。接下来，执行录制的 Flight 测试脚本。

① 打开录制的 Flight 测试脚本。

② 设置运行选项。选择 Tool→Options 命令，打开设置选项对话框，选择 Run 选项卡，如图 15.19 所示。

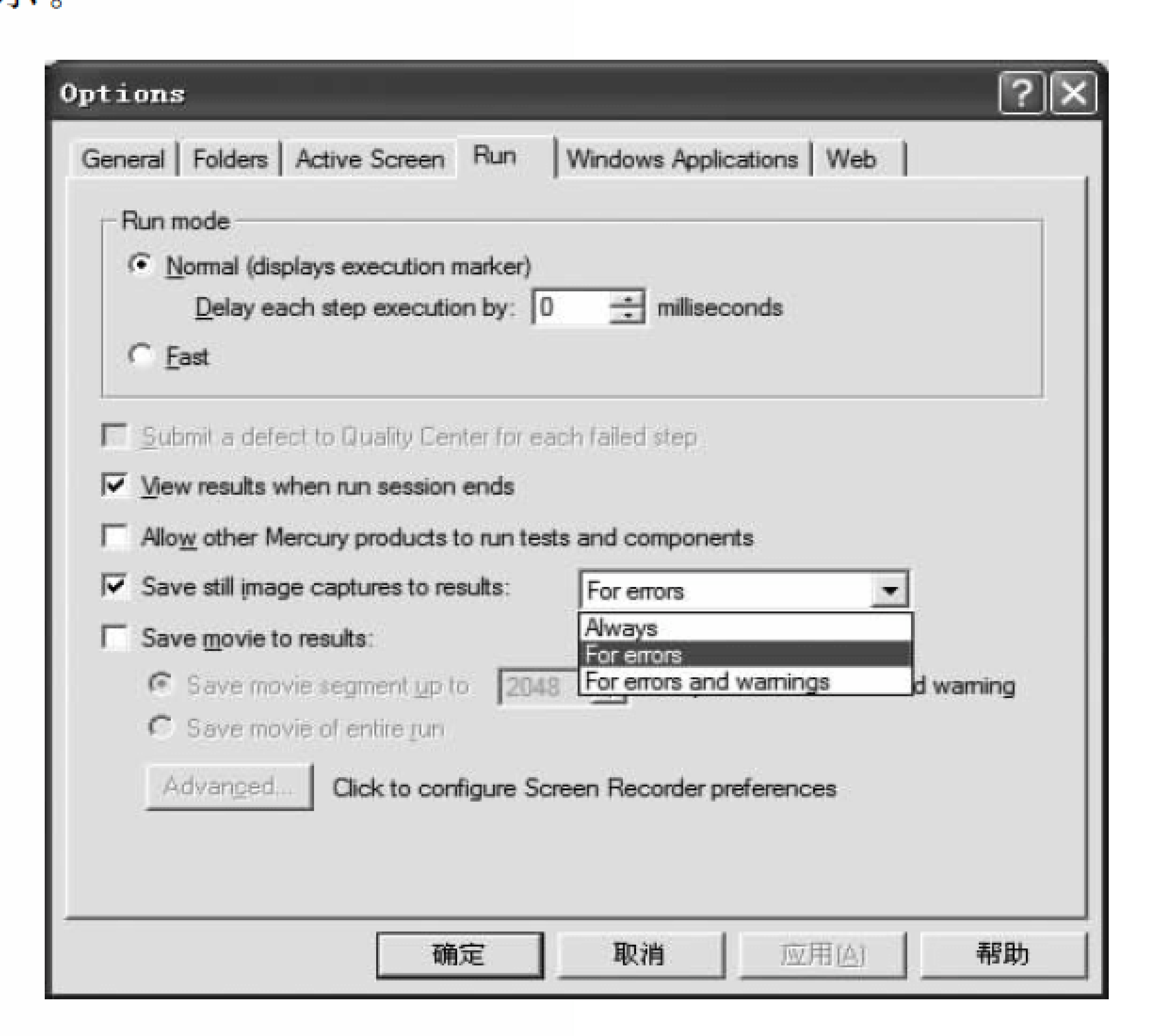


图 15.19　QTP Options 设置

③ 在工具条上单击 Run 按钮，打开 Run 对话框，如图 15.20 所示。

④ 可以看到 QTP 按照在脚本中录制的操作，一步一步地运行测试，在 QTP 的 Keyword View 中会出现一个黄色的箭头，指示目前正在执行的测试步骤。

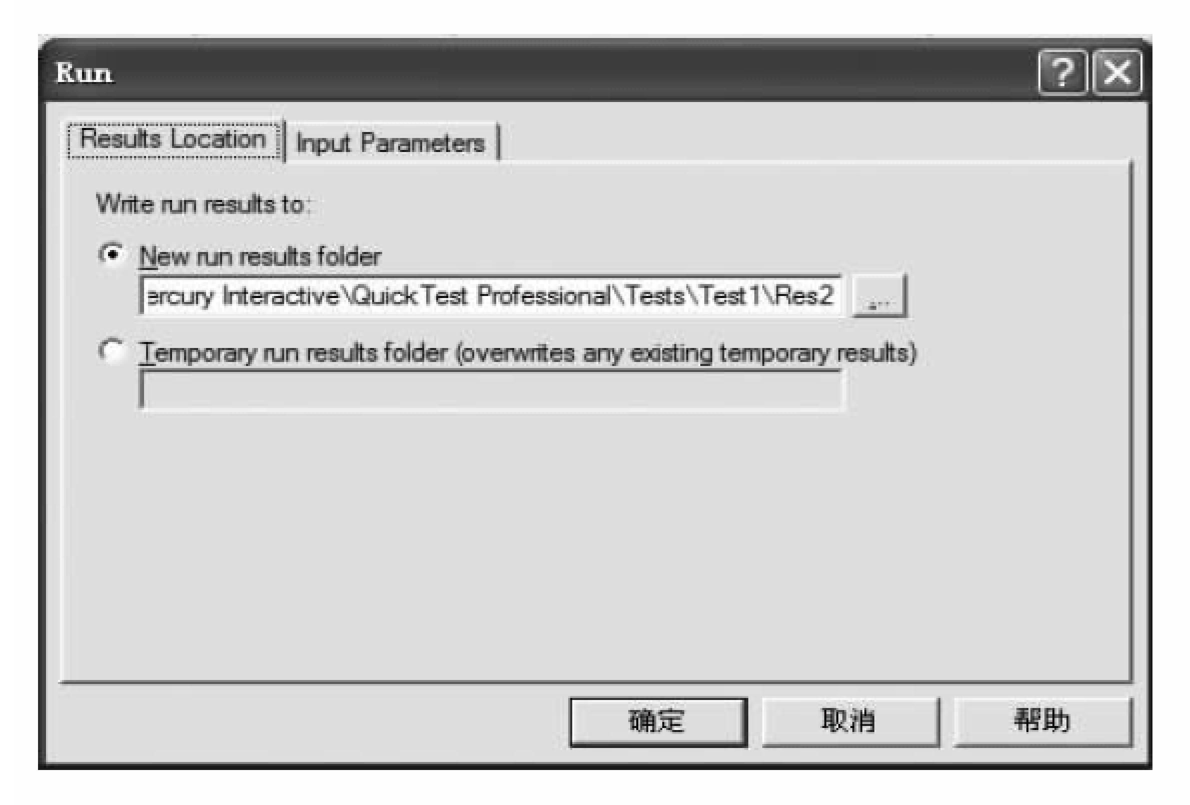


图 15.20 QTP Run 设置

⑤ 如果在执行测试的时候出现错误，会显示一个错误信息对话框。

(4) 查看测试结果

在测试执行完成后，QTP 会自动显示测试结果窗口，如图 15.21 所示。

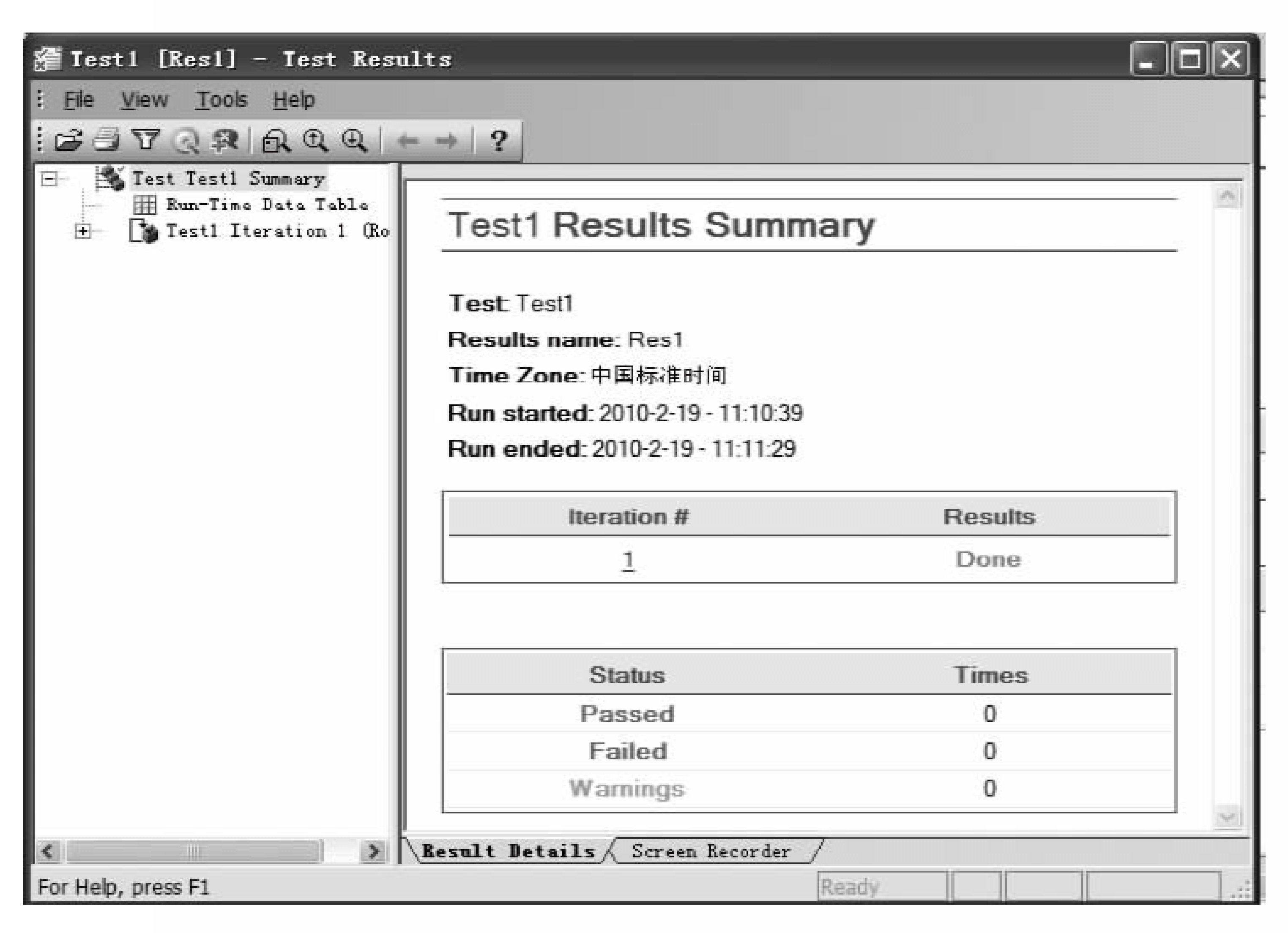


图 15.21 QTP 测试结果界面

在这个测试结果窗口中分两个部分显示测试执行的结果。

- 左边显示 Test results tree，以阶层图标的方式显示测试脚本所执行的步骤。可以选择“+”检查每一个步骤，所有的执行步骤都会以图示的方式显示。可以设定 QTP 以不同的资料执行每个测试或某个动作，每执行一次反复称为一个迭代，每一次迭代都会被编号。
- 右边则是显示测试结果的详细信息。在第一个表格中显示哪些迭代是已经通过

的，哪些是失败的。第二个表格是显示测试脚本的检查点，哪些是通过的，哪些是失败的，以及有几个警告信息。

在树视图中展开 Flight Iteration 1（Row 1）→Action1 Summary→Welcome MercuryTours→Find a Flight：Mercury，选择“fromPost：Select”—“New York”。

在这个测试结果窗口中显示三个部分，分别是：

- 左边是 Test results tree：展开树视图后，显示了测试执行过程中的每一个操作步骤。选择某一个测试步骤，会在右边区域显示相应的信息。
- 右上方是 Test results detail：对应当前选中的测试步骤，显示被选取测试步骤执行时的详细信息。
- 右下方是 Active Screen：对应当前选中的测试步骤，显示该操作执行时应用程序的屏幕截图。当选中 test results tree 上的网页图示，会在 Active Screen 中看到执行时的画面，如图 5.22 所示。

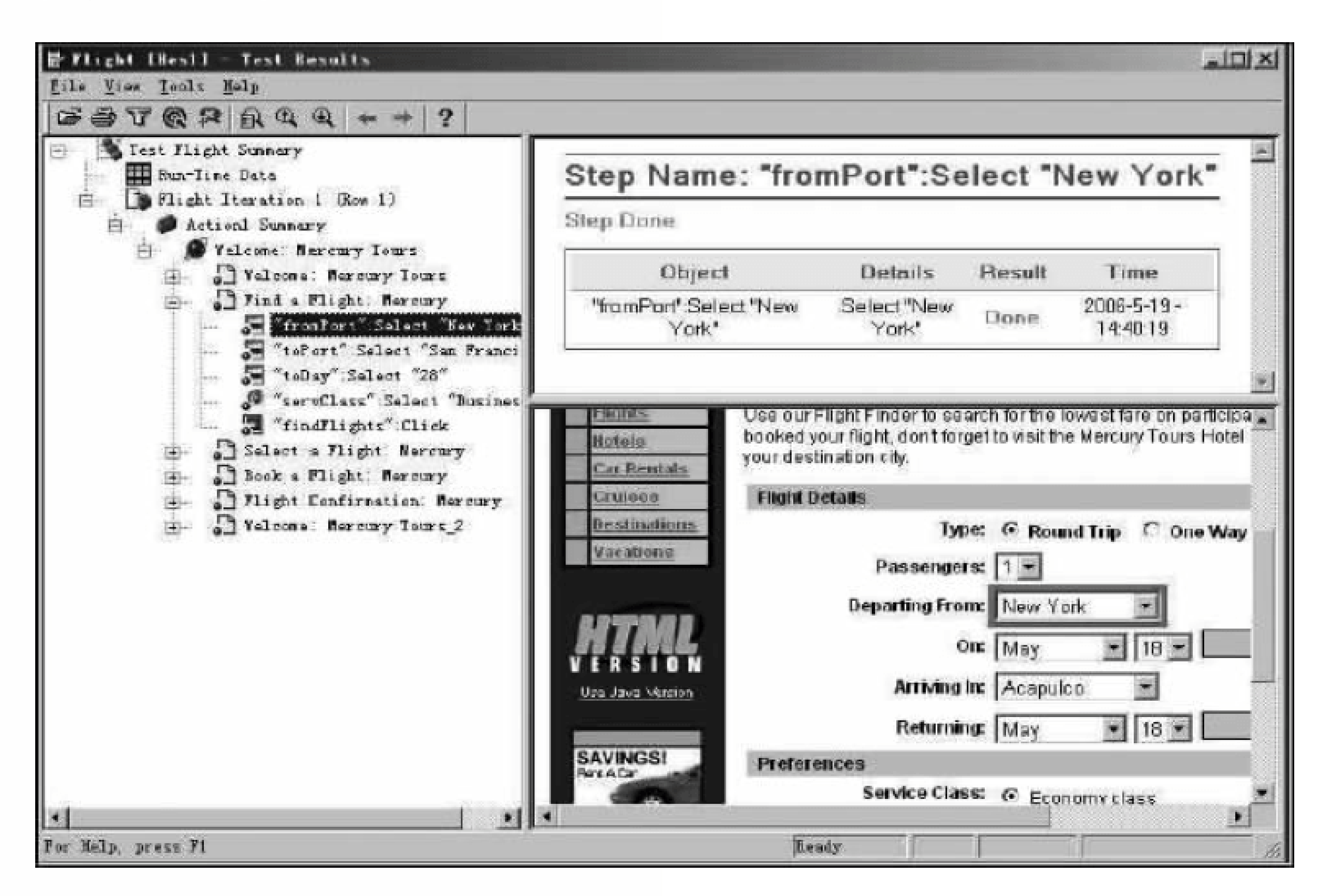


图 15.22　QTP 测试结果界面

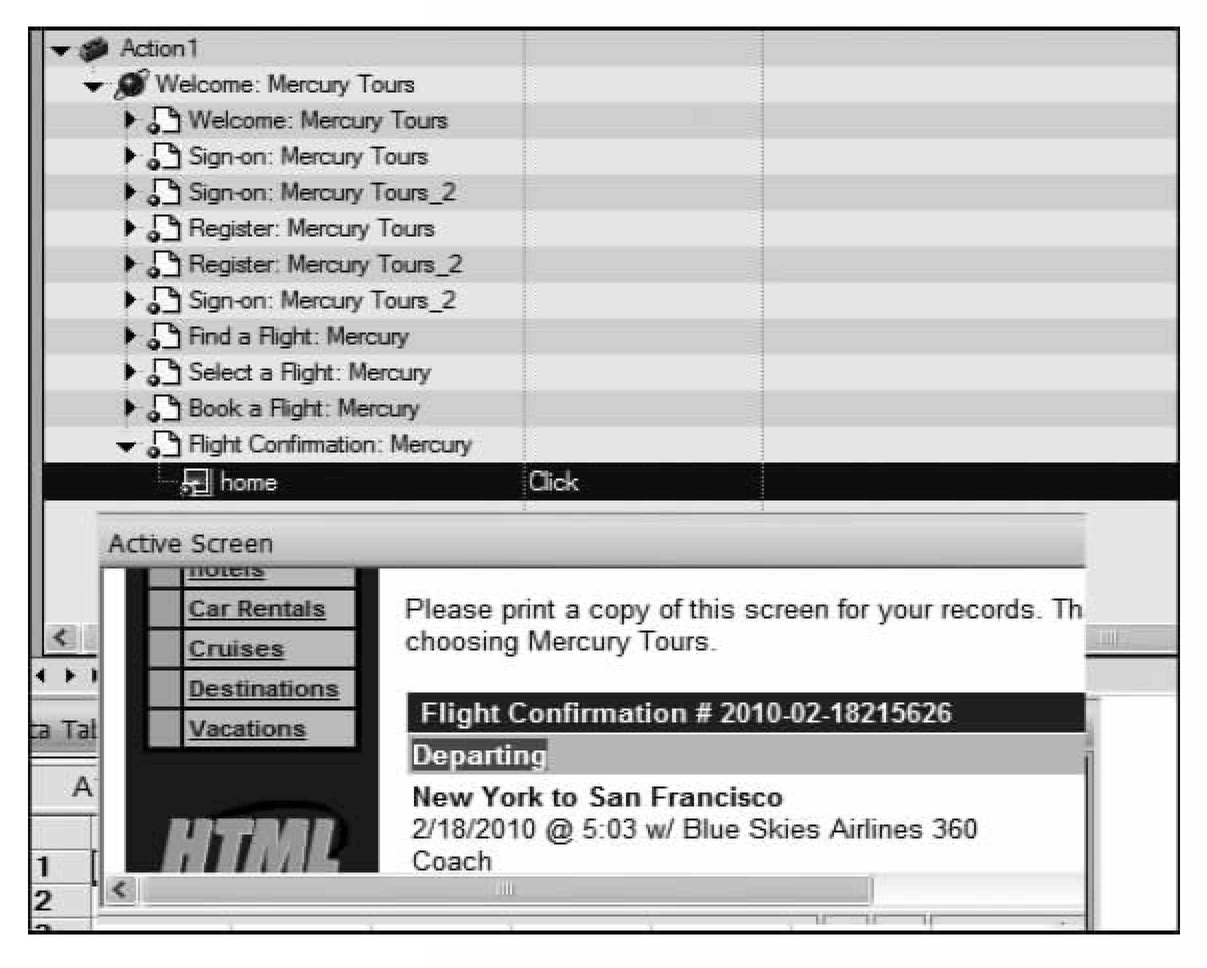
（5）设置检查点

“检查点”是将指定属性的当前值与该属性的期望值进行比较的验证点。这能够确定网站或应用程序是否正常运行。当添加检查点时，QTP 会将检查点添加到关键字视图中的当前行并在专家视图中添加一条“检查检查点”语句。运行测试或组件时，QTP 会将检查点的期望结果与当前结果进行比较。如果结果不匹配，检查点就会失败。可以在“测试结果”窗口中查看检查点的结果。

打开 Flight 测试脚本，将脚本另存为 Checkpoint 测试脚本。在 Checkpoint 测试脚本中创建 1 个文字检查点。检查在 Flight Confirmation 网页中是否出现 New York?

① 确定要建立检查点的网页：展开 Action1→Welcome：Mercury Tours，选择 Flight Confirmation：Mercury 页面，在 Active Screen 会显示相应的页面。

② 建立文字检查点：在 Active Screen 中选择在 Departing 下方的 New York。右击文字，选取 Insert Text Checkpoint，打开 Text CheckpointProperties 对话框。当 Checked Text 出现在下拉式清单中时，在 Constant 字段显示的就是选取的文字，作为 QTP 在执行测试脚本时所要检查的文字，如图 15.23 和图 15.24 所示。



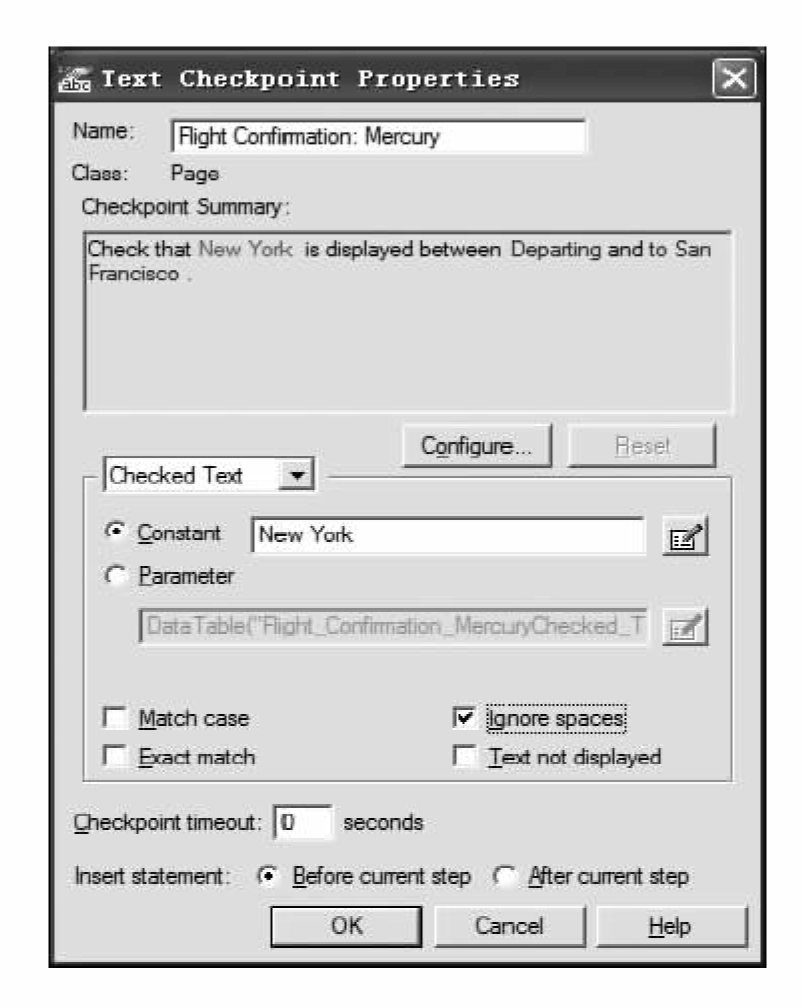



图 15.23 QTP 加入检查点

图 15.24 QTP 检查点设置

③ 单击 OK 按钮，关闭窗口。QTP 会在测试脚本上加上一个文字检查点，这个文字检查点会出现在 FlightConfirmation：Mercury 网页下方。

④ 工具栏上点击“Save”保存脚本。

(6) 执行测试。

① 打开录制的 Flight 测试脚本。

② 运行脚本。

(7) 查看测试结果

在 test results tree 中展开 Checkpoint Iteration 1 (Row 1)→Action1 Summary→Welcome:Mercury Tours→Flight Confirmation：Mercury，并选择 Checkpoint "New York"。如图 15.25 所示，由于文字检查点的实际值与预期值相同，所以检查点的结果为 Passed。

(8) 参数化设置

由于在测试脚本中，纽约是个常数值，也就是说，每次执行测试脚本预订机票时，出发地点都纽约，现在，将测试脚本中的出发地点参数化，这样，执行测试脚本时就会以不同的出发地点去预订机票了。

① 打开 Checkpoint 测试脚本，将脚本另存为 Parameter，然后选择要参数化的文字：在视图树中展开 Action1→Welcome：Mercury Tours→Find a Flight：Mercury，如图 15.26 所示。

② 在视图树中选择 fromPort 右边的 Value 字段，然后再单击参数化图标，开启

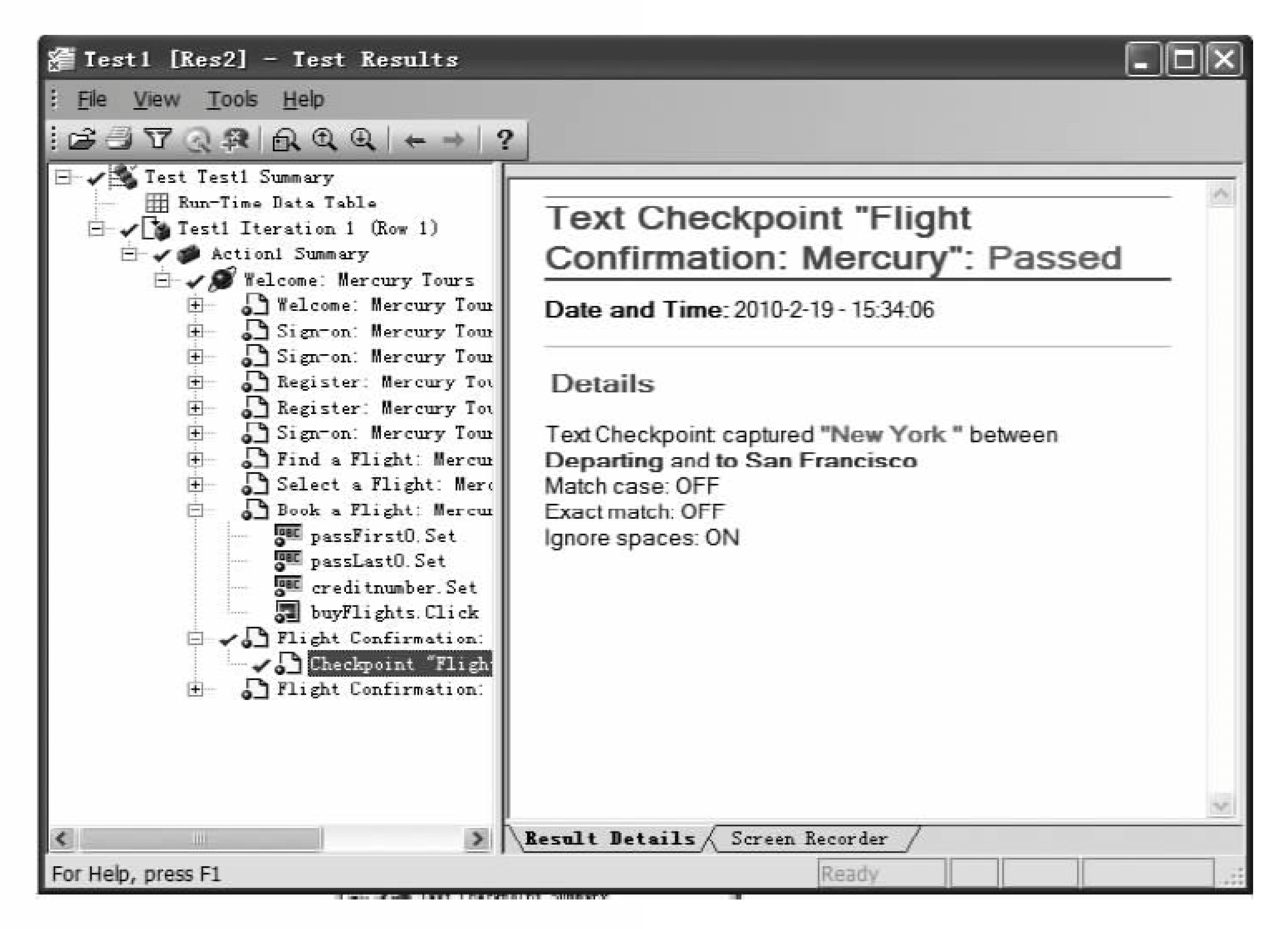


图 15.25 QTP 检查点测试结果界面

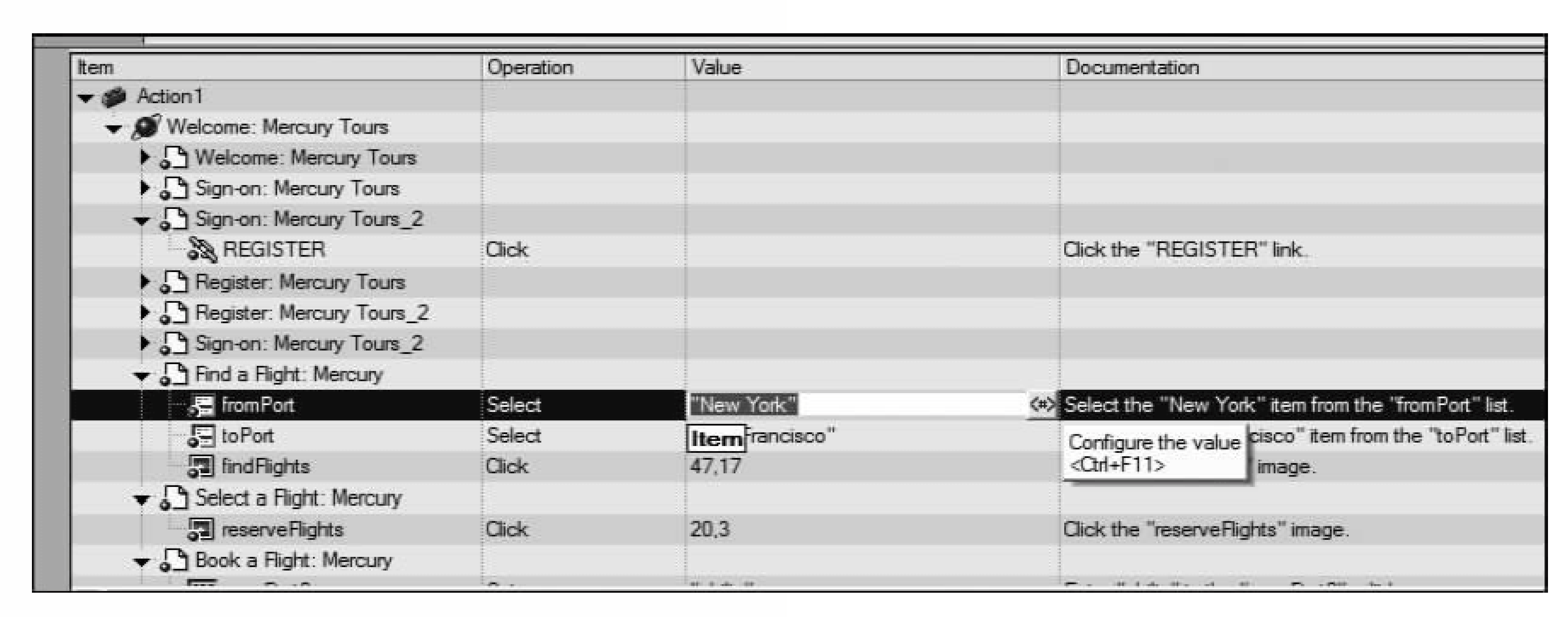


图 15.26 QTP 脚本管理加入参数化

Value Configuration Options 对话框。

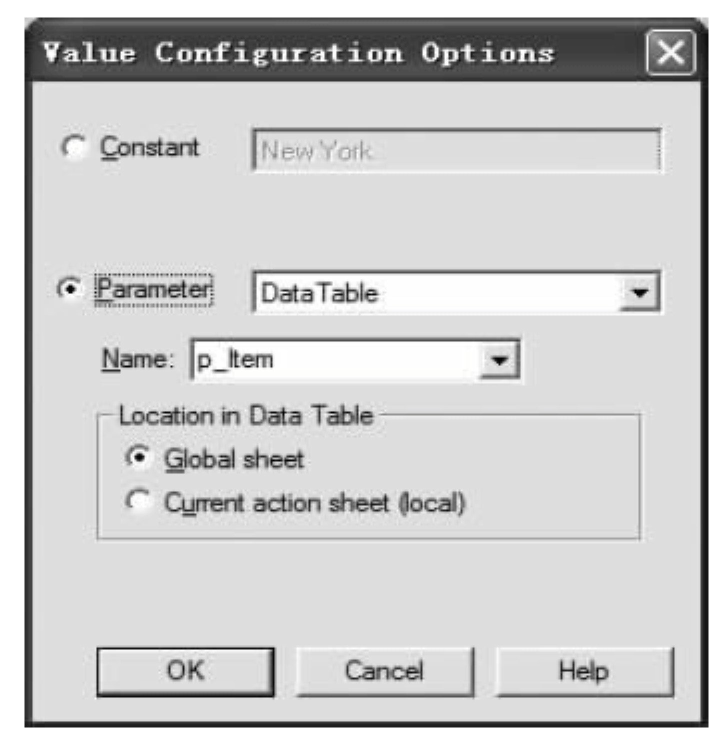


图 15.27 QTP 操作工具栏

③ 设置要参数化的属性，选择 Parameter 选择项，这样就可以用参数值来取代 NewYork 这个常数了，在参数中选择 Data Table 选项，这样这个参数就可以从 QTP 的 Data Table 中取得，将参数的名字改为 p_Item，如图 15.27 所示。

④ 单击 OK 按钮确认，QTP 会在 Data Table 中新增 departure 参数字段，并且插入了一行 New York 的值，New York 会成为测试脚本执行使用的第一个值。

⑤ 参数化以后可以看到树视图中的变化，如图 15.28 所示。在 departure 字段的第二行，第三行分

别输入：Portland、Seattle。

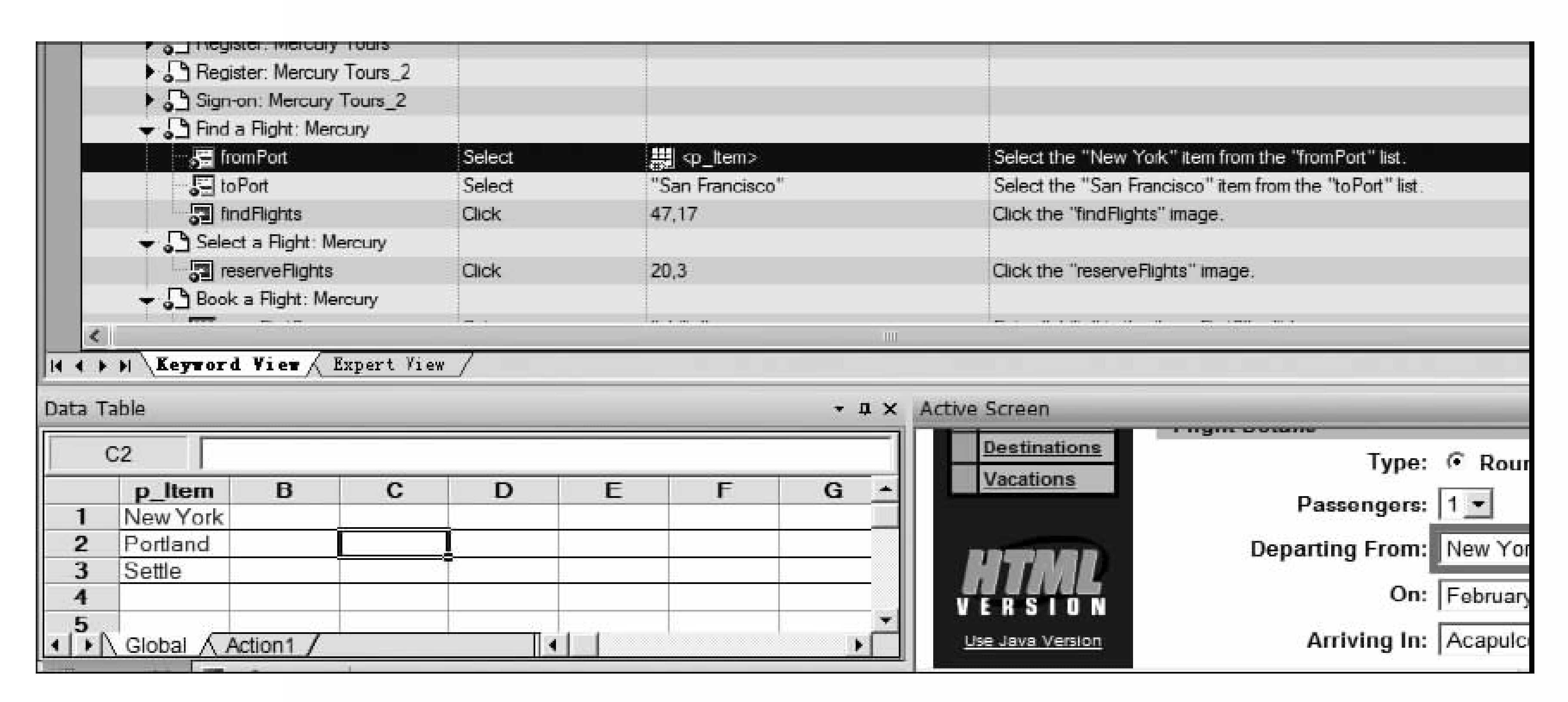


图 15.28　QTP 脚本界面-参数化后

⑥ 保存测试脚本。

⑦ 修正受到参数化影响的步骤：当测试步骤被参数化以后，有可能会影响到其他的测试步骤也要参数化。

⑧ 修正文字检查点，首先在树视图中，展开 Action1 Summary→Welcome：Mercury Tours→FlightConfirmation：Mercury 页面，然后右击，选择 Checkpoint Properties，打开 TextCheckpoint Properties 对话框，如图 15.29 所示。

⑨ 在 Checked Text 的 Constant 字段中显示为 New York，表示测试脚本在每次执行时，这个文字检查点的预期值都为 New York。选择 Parameter，单击旁边的 ParameterOptions 按钮，打开 Parameter Options 对话窗口：在参数类型选择框选择 Data Table 选项，在名字选择框选择 p_Item 选项，指明这个文字检查点使用 p_Item 字段中的值当成检查点的预期值，如图 15.30 所示。

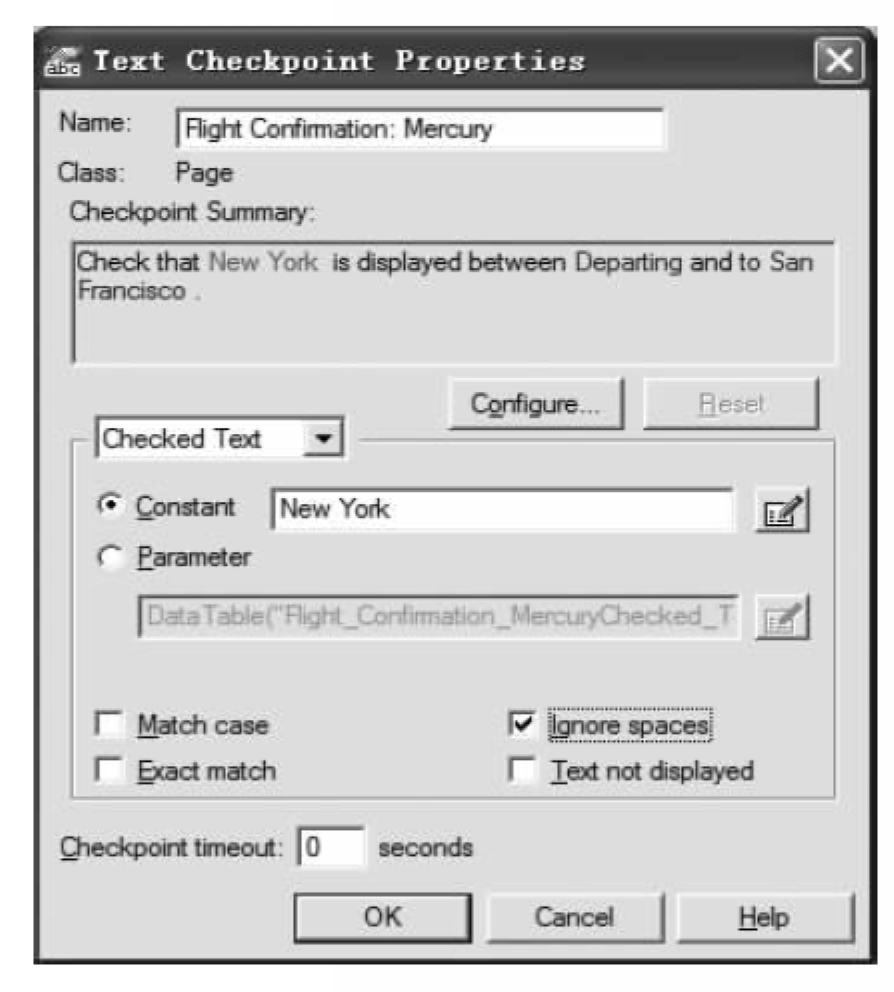


图 15.29　QTP 修正文字检查点(1)

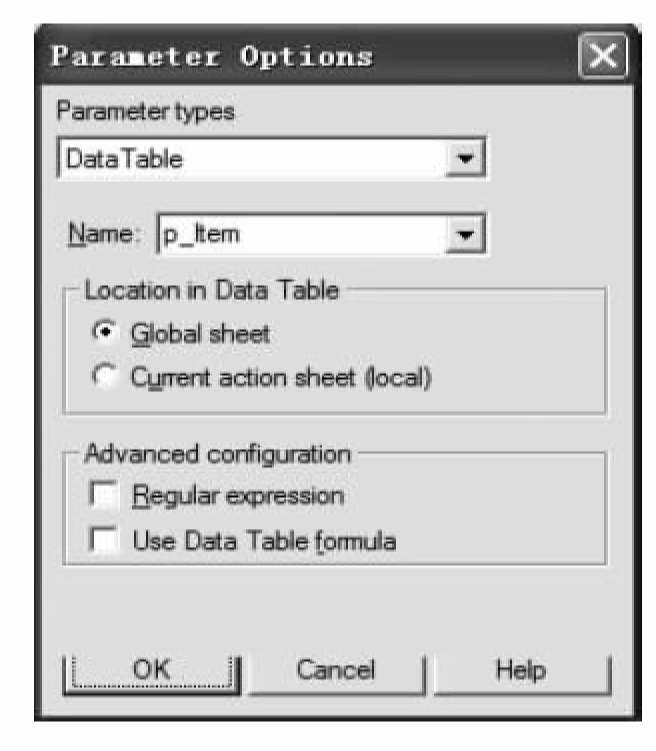


图 15.30　QTP 修正文字检查点(2)

（9）执行测试

参数化测试脚本后，运行 Parameter 测试脚本。QTP 会使用 Data Table 中 p_Item 字段值，执行三次测试脚本。

（10）查看测试结果

在树视图中，展开 Parameter Iteration2 → Action1 Summary → Welcome MercuryTours→Flight Confirmation：Mercury，选择 Checkpoint "New York"，显示如图 15.31 所示。

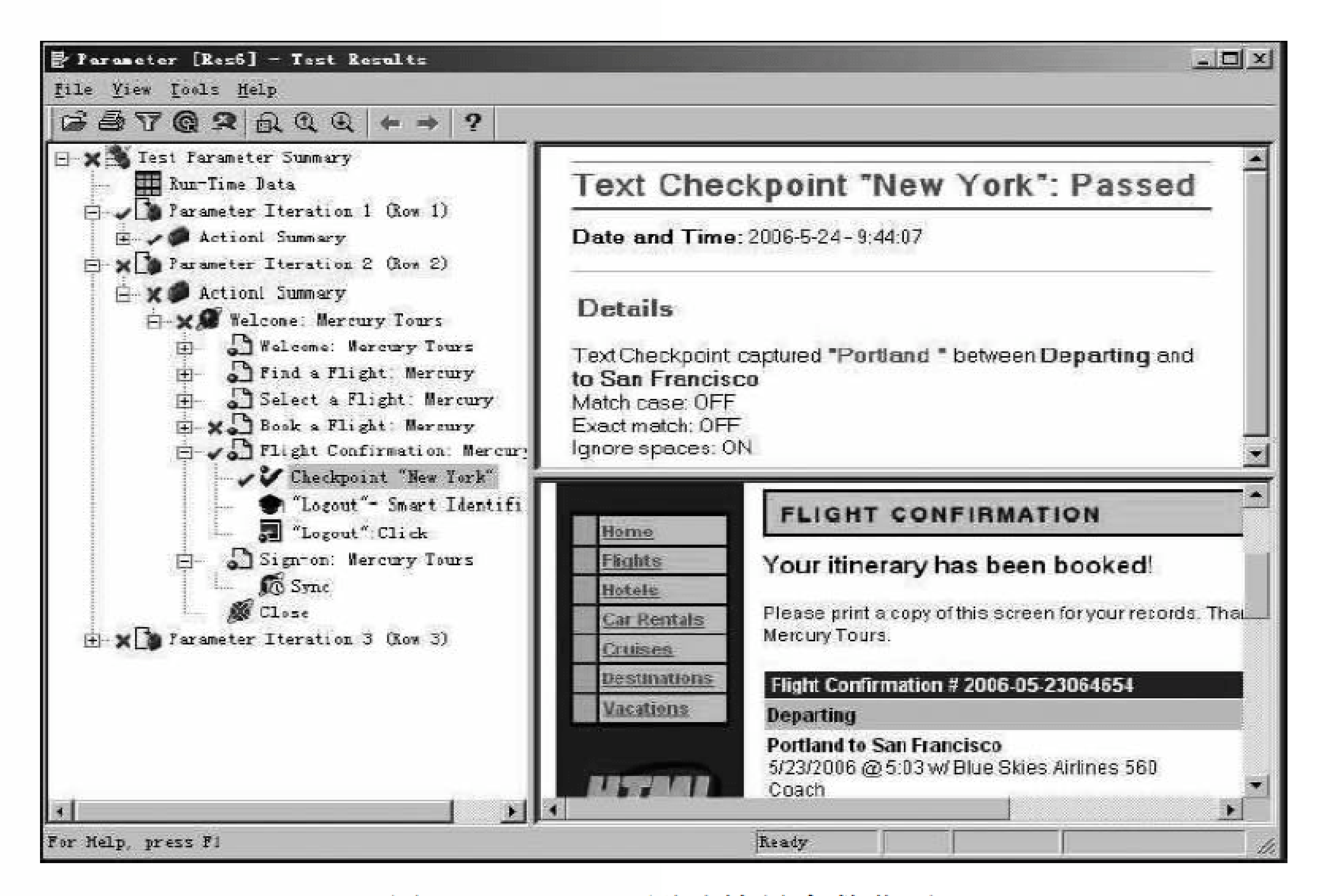
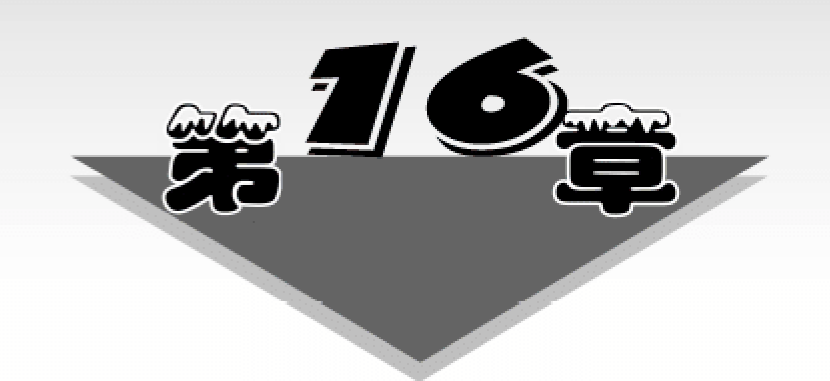


图 15.31　QTP 测试结果参数化后

在检查点 Details 窗口中，显示 Portland 为预期记过同时也是实际的值，所以文字检查点为通过。同时也可以看到在下方的 Application 窗口中，显示机票的出发地点也是 Portland。虽然每次执行时，文字检查点的结果是通过的，但是第二次与第三次的执行结果仍然为失败。这是因为出发地点的改变，造成在表格检查点中的机票价钱改变，导致表格检查点失败。



嵌入式软件测试工具

16.1 Logiscope 简介

Logiscope 是嵌入式软件测试工具集，用于软件开发、代码评审、单元/集成测试、系统测试以及软件维护阶段等。Logiscope 针对编码、调试和维护，帮助代码评审和动态覆盖测试。Logiscope 采用 Halstead 质量模型，从软件的编程规则、静态特征和动态测试覆盖等多个方面，检查和评估软件质量。

Logiscope 在软件开发的各个阶段都各自的功能，具体如下所示。

1. 用于开发阶段

根据所选的规则对源代码进行验证，指出所有不符合编程规则的代码，并提出改进源代码的解释和建议。通过将被评价的软件与规定的质量模型进行比较，用图形形式显示软件质量的级别，对度量元素和质量模型不一致的地方作出解释并提出纠正的方法。

2. 用于测试阶段

Logiscope 对指令、逻辑路径和调用路径的覆盖测试，对安全关键软件进行 MC/DC 的覆盖测试。TestChecker 产生每个测试的测试覆盖信息和累计信息，用直方图显示覆盖比率，并根据测试运行情况实时在线更改，随时显示新的测试所反映的测试覆盖情况。TestChecker 允许所有的测试运行依据其有效性进行管理。

在执行测试期间，当测试策略改变时，综合地运用 TestChecker 检测关键因素以提高效率。将 TestChecker 与 Audit 配合使用能够帮助用户分析未测试的代码。用户可以显示所关心的代码，并通过对执行未覆盖的路径的观察得到有关的信息。信息以图形和文本的形式提交，并在其间建立导航关联。TestChecker 管理系统声明新的测试、生成有关文档、定义启动命令以及自动执行的方法。

3. 用于维护阶段

Logiscope 可以大大地减少对未知系统的理解所需的时间。Audit 将应用系统的框架以文件形式和图的形式进行可视化。函数的逻辑结构以控制流图的形式显示。在控制流图上选定一个节点，即可得到相对应的代码。可以在不同的抽象层上对应用系统进行分析，不同层次间的航，促进对整体的理解。

4. 对嵌入式领域的支持

由于嵌入式系统软件的开发是用交叉编译方式进行，因此，其测试较为麻烦。在目标机上，不可能有多余的空间记录测试的信息。必须实时地将测试信息通过网线/串口传到宿主机上，并实时在线地显示。因此，对源代码的插装和目标机上的信息收集与回传成为问题的关键。

5. 软件文档和测试文档的自动生成

Logiscope 提供了文档自动生成工具。使用者可以将代码评审的结果和动态测试情况实时生成所要求的文档，这些文档忠实地记录代码的情况和动态测试的结果。文档的格式可以根据用户的需要定制。

16.2 Logiscope 三大功能

将 Logiscope 的 license(注册)服务安装在服务器上，在客户端上使用 Logiscope，具体步骤如下所示：

(1) 安装 Logiscope，只需按照安装程序的提示操作即可。

确保安装过程正确无误，并保证安装的机器可以和提供 Logiscope 的 license 服务的机器联网后，进行下面的操作。

(2) 在计算机上，右击“我的电脑”图标，选择“属性”菜单项。在弹出的对话框中选中“高级”选项卡，单击“环境变量”按钮，在弹出的对话框中，选中“系统变量”列表框中的 LM_LICENSE_FILE 这一项。单击“编辑”按钮，如图 16.1 所示。

(3) 此时会弹出新的对话框。在该对话框的“变量值”编辑框中，添入 Logiscope 为提供的 license 号，以此来取代原来的值，如图 16.2 所示。



图 16.1 系统特性环境变量



图 16.2 系统特性编辑环境变量

(4) 依次单击“确定”按钮,退出各个对话框。

至此,Logiscope 安装与配置完成。下面开始介绍 Logiscope 的 Audit、RuleChecker、TestChecker 三项功能。

16.2.1 使用 Audit

Audit 翻译为审查、检查,用于审查代码的质量。使用 Audit 来审查代码的质量分为两个步骤:首先是建立被测程序的 Audit 项目,然后是分析 Audit 给出的质量审查结果。

生成被测程序的 Audit 项目有如下两种方法。

(1) 第一种方法:Logiscope studio 中建立 Audit 项目。

① 在开始菜单中,启动 Logiscope studio,如图 16.3 所示。

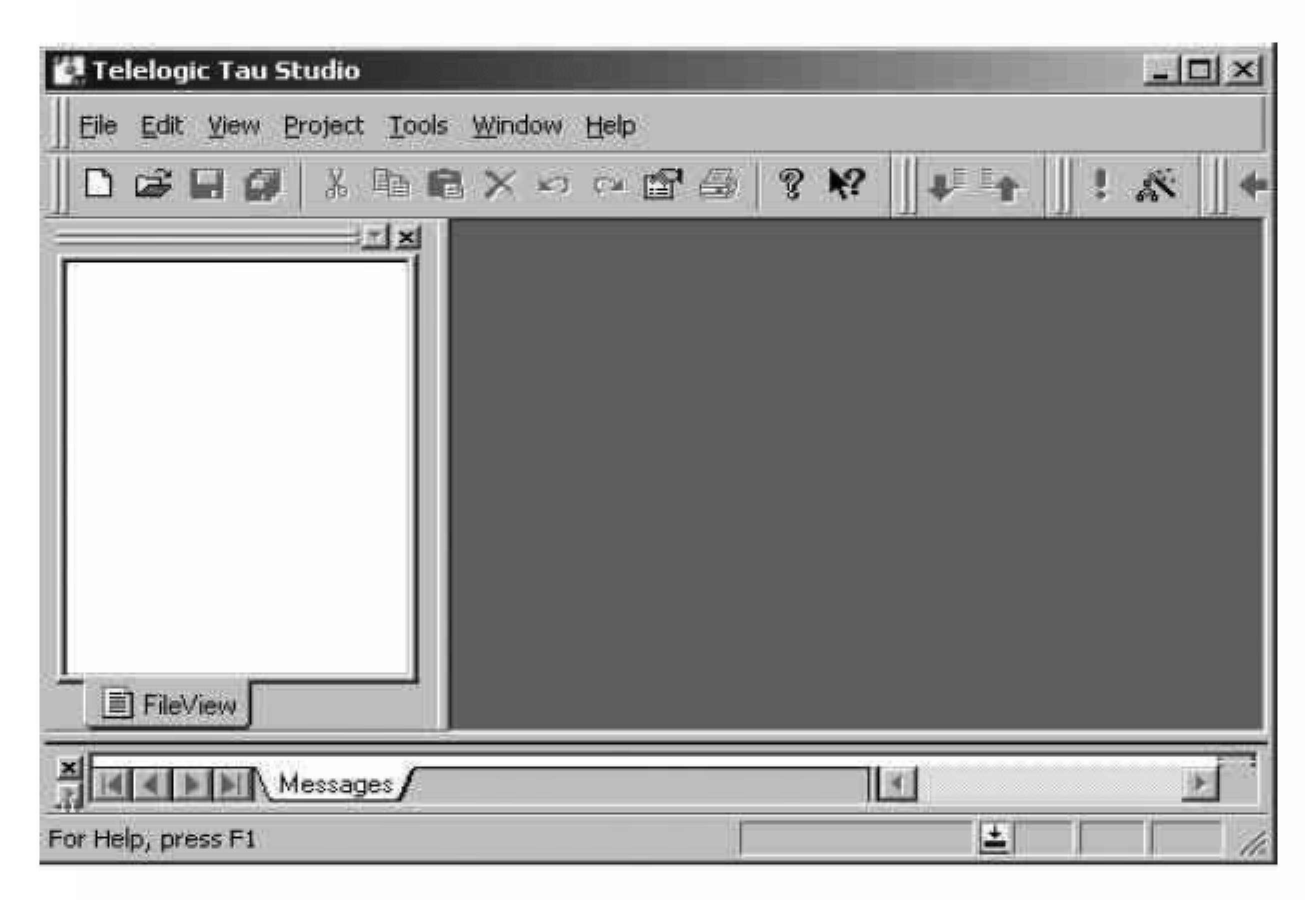


图 16.3 Logiscope studio 环境

② 选择 File→New 命令,弹出如图 16.4 所示的对话框。

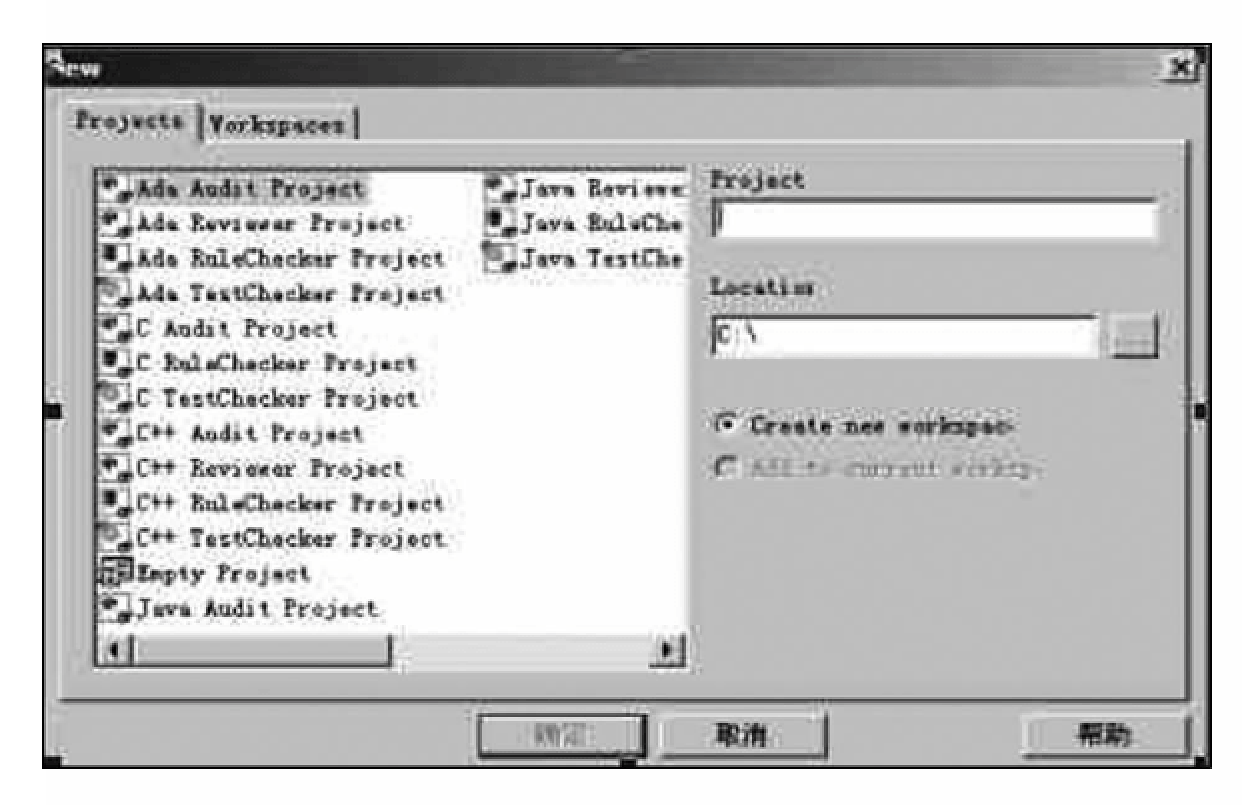


图 16.4 新建项目

在对话框中,选中 Project 选项卡,在列表框中选择 C++ Audit Project 这一项,然后

在 Project 编辑框中添入要建立的这个 Audit 项目的名字，再在 Location 编辑框中选择一个存放将要生成的 Audit 项目的文件目录。最后单击“确定”按钮。

③ 此时会弹出如图 16.5 所示的对话框。



图 16.5　新建项目向导

在对话框的 Application root 编辑框中，添入所要检测的源程序文件所在文件夹的路径，其他均采用默认设置，然后单击“下一步”按钮。

④ 弹出如图 16.6 所示的对话框。



图 16.6　新建项目向导

在该对话框中，使 Choose a parser 组合框保持“MFC”默认选项，在 Choose a quality 编辑框中，添入所设计的质量模型文件的存放路径（Logiscope 默认选择的是 LogiscopeHOME\Logiscope\Ref\Logiscope.ref 下的这个质量模型文件），Logiscope 要依照该文件对被审查的代码进行检测。在 Choose a Logiscope 编辑框中为生成的 Logiscope 中间结果文件选择一个存放路径，一般使用当前提供的默认路径即可，最后单击“下一步”按钮。

⑤ 弹出如图 16.7 所示的对话框。

这个对话框汇报将要生成的 Audit 项目的相关情况。单击“完成”按钮。至此，生成



图 16.7 新建项目向导

了一个 Audit 项目，显示窗口如图 16.8 所示。

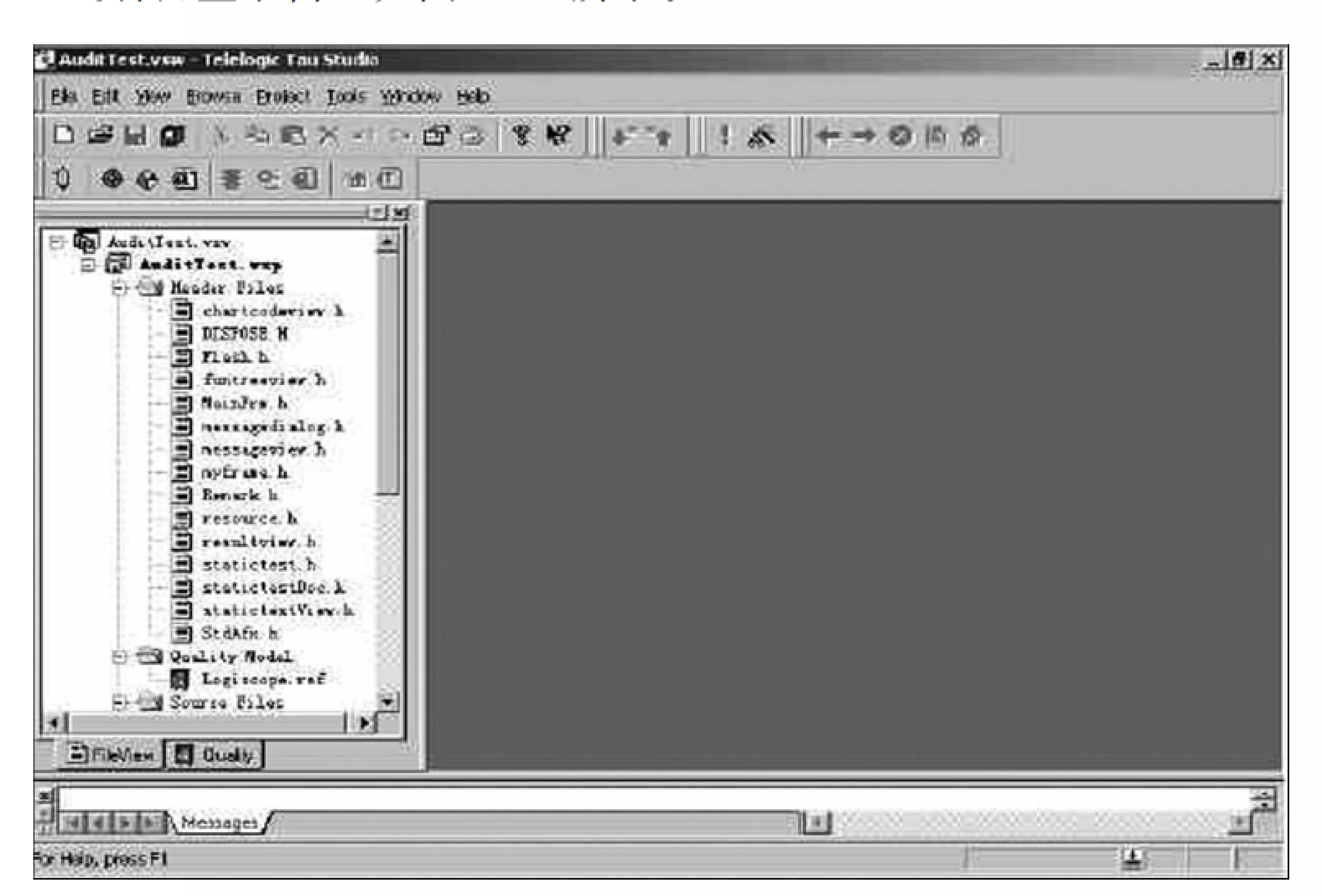


图 16.8 新建项目结束

在该窗口中，选择 Project→Build 命令，Audit 开始对被测代码进行检测。Build 执行结束后，代码质量的检测结果也就产生了，进行保存。

(2) 第二种方法：在 Visual Studio 中建立 Audit 项目。

启动 VC6.0，打开要检测的项目，选择 Tools→Build Quality Results 菜单项，Build 结束后，选择 Tools→Viewer 菜单项，此时，Logiscope 被启动，Audit 对代码质量的检测结果会显示出来。

选择 Browse→Quality→Factor Level 菜单项，Logiscope 会显示 Audit 对所检测源程序质量水平的评价结果，评价结果包括系统的质量、类的质量、函数的质量，如图 16.9 所示。

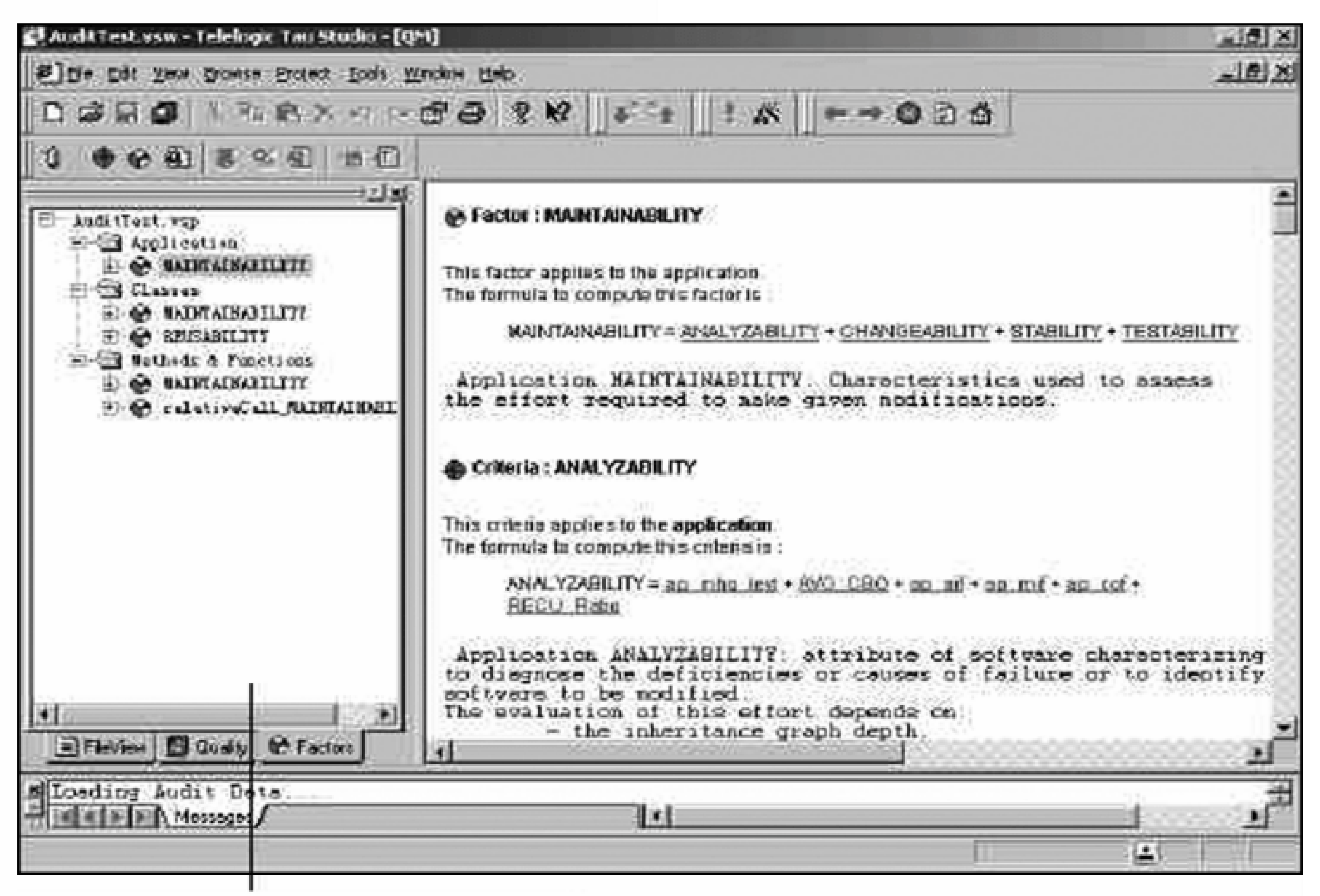


图 16.9　质量水平

选择 Browse→Quality→Criteria Level 命令，Logiscope 会显示 Audit 对所测源程序的各项质量标准的检测结果，具体包括：系统的质量标准、类的质量标准、函数的质量标准，如图 16.10 所示。

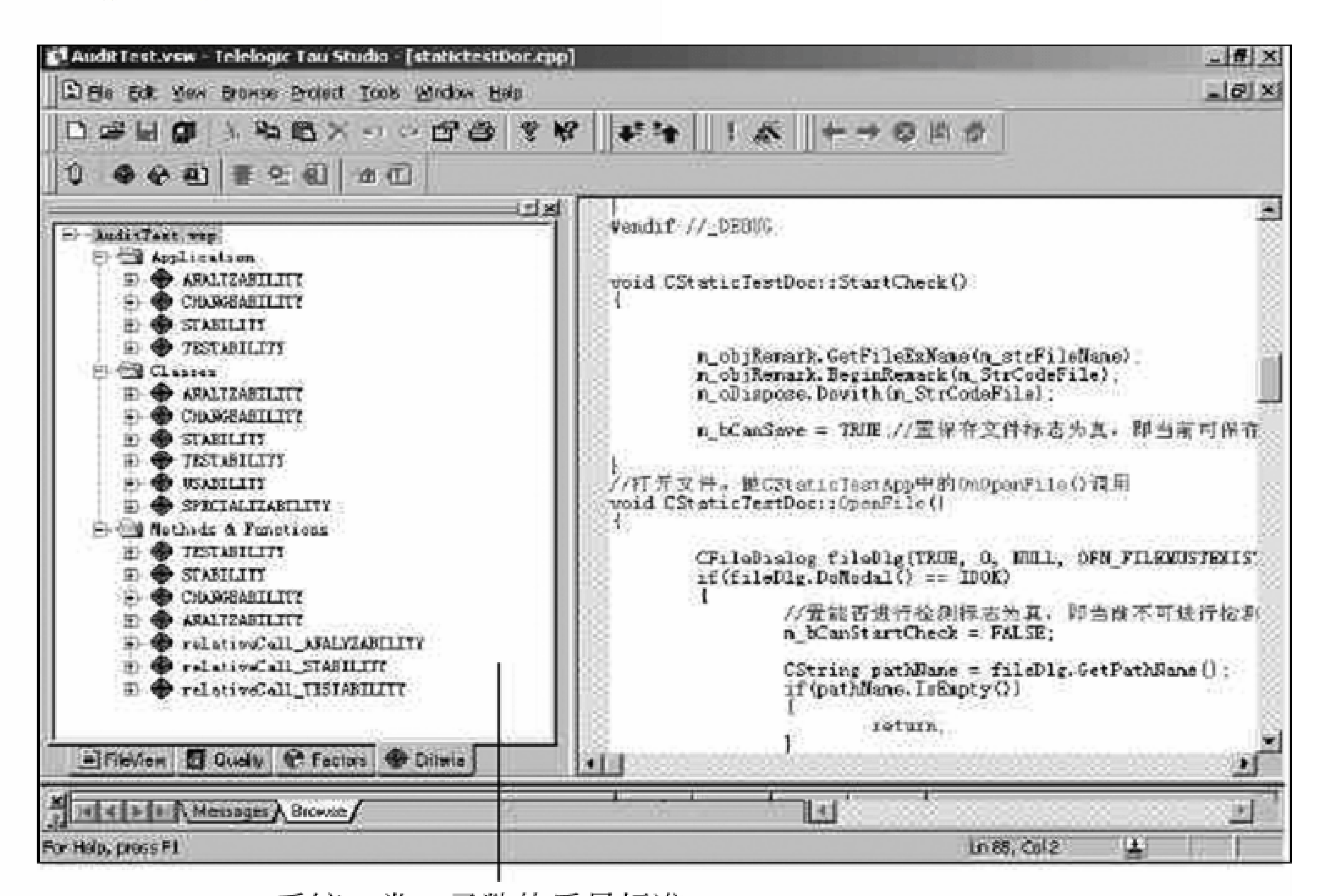


图 16.10　质量标准

选择 Browse→Quality→Quality Report 命令，可生成网页风格的系统质量评价报告。如图 16.11 所示。



图 16.11　质量评价报告

关于系统、类、函数在质量度量级上的检测信息，需要选择 Project→Start Viewer 命令，通过启动 Logiscope Viewer 来进行查看。Logiscope Viewer 被启动后，界面如图 16.12 所示。

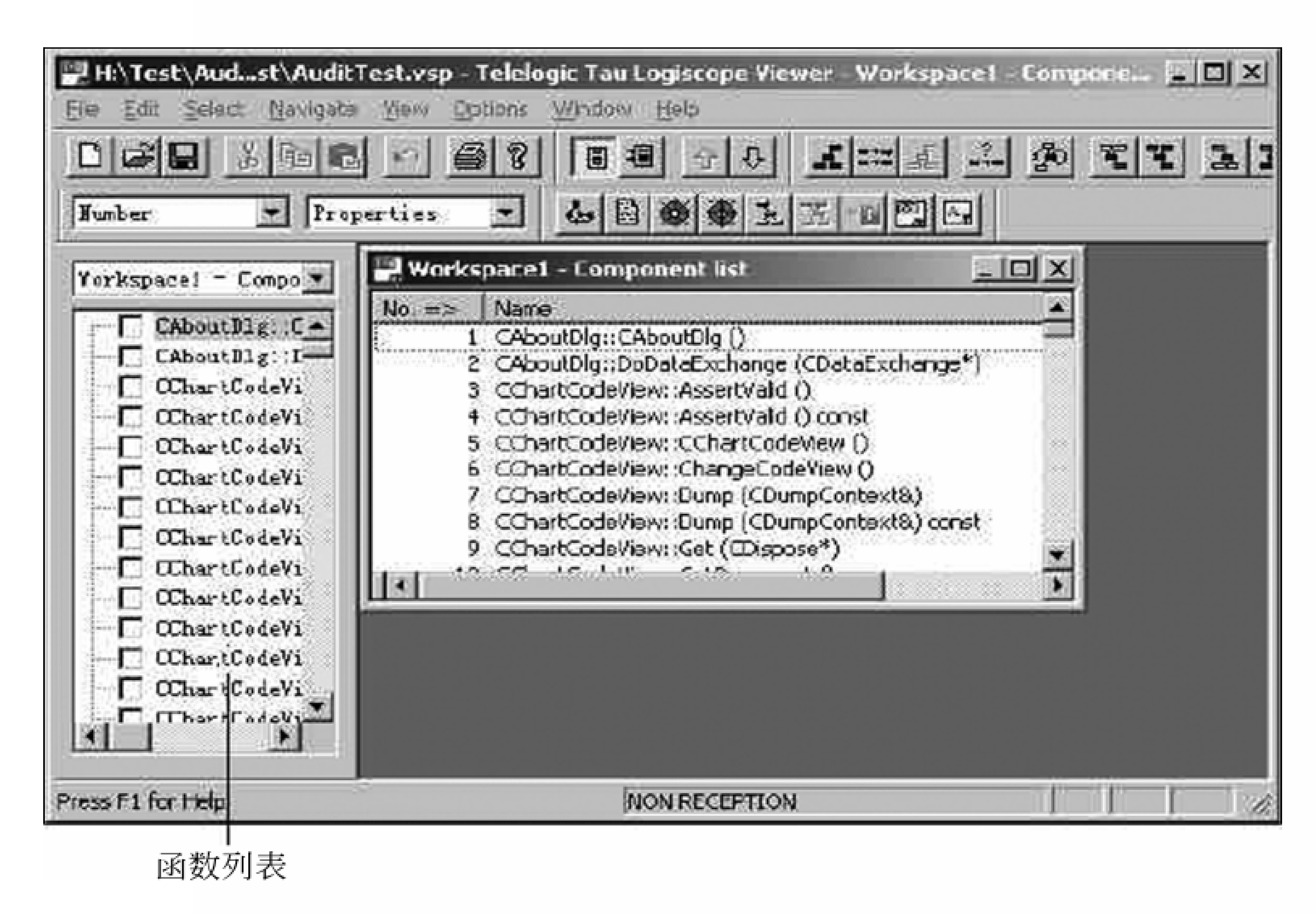


图 16.12　Viewer 界面

Viewer 中的列表控件中，显示了系统中的全部函数。选中某个函数后，通过点击下面这个工具条上的按钮，可以查看 Audit 提供的对函数的各种分析信息。工具条及工具

条上各个按钮的功能如图 16.13 所示。

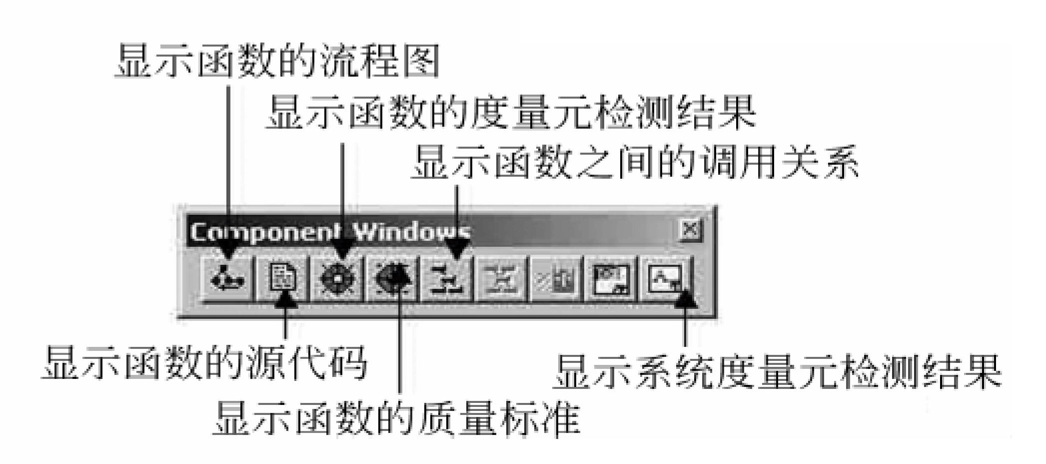


图 16.13　函数分析信息工具条

单击函数流程图按钮，会显示函数的流程图 16.14 所示。

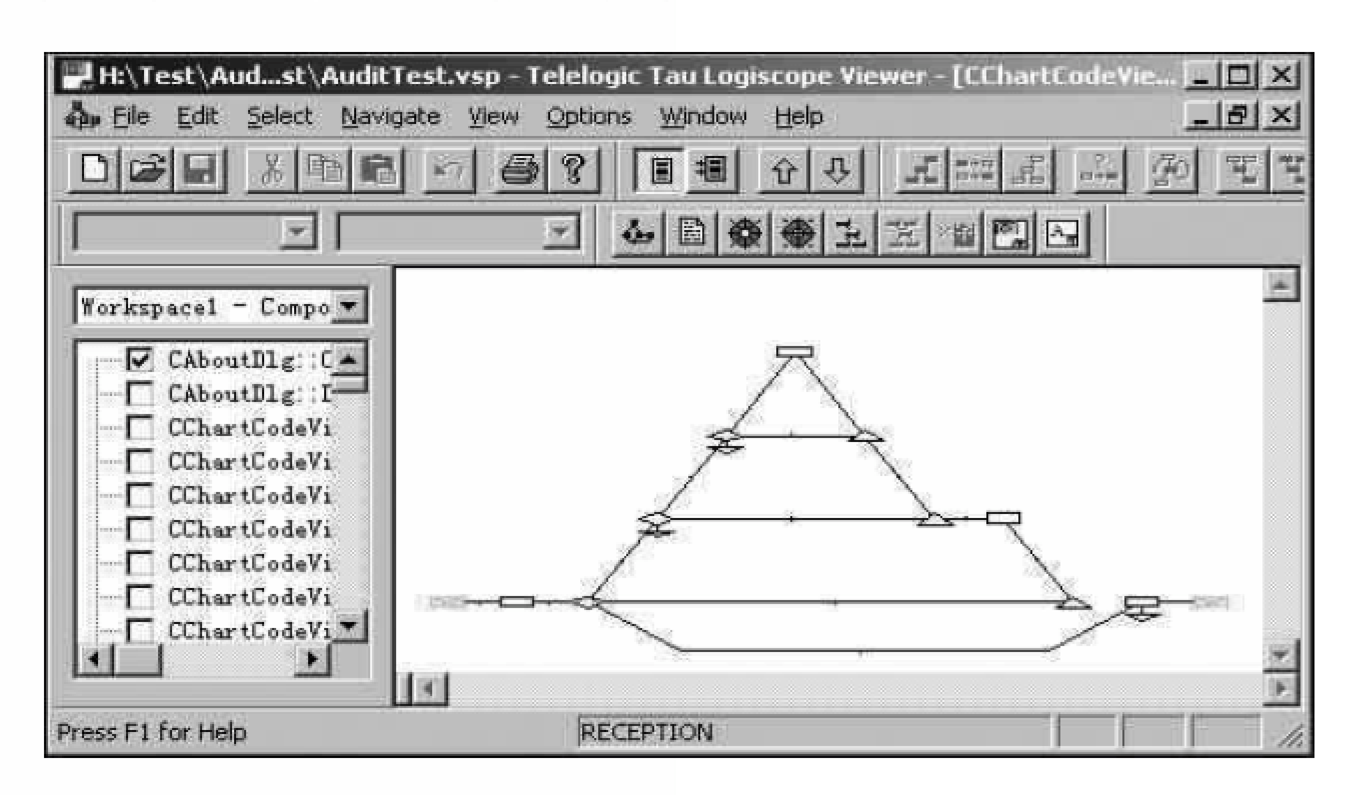


图 16.14　函数流程图

选择 Options→Full Reduction 或 Options→Step-by-Step Reduction 命令，可对流程图进行结构化的转换，选择 Options→Initial 命令还原。单击函数度量元按钮，会显示函数的度量元检测结果，如图 16.15 所示。

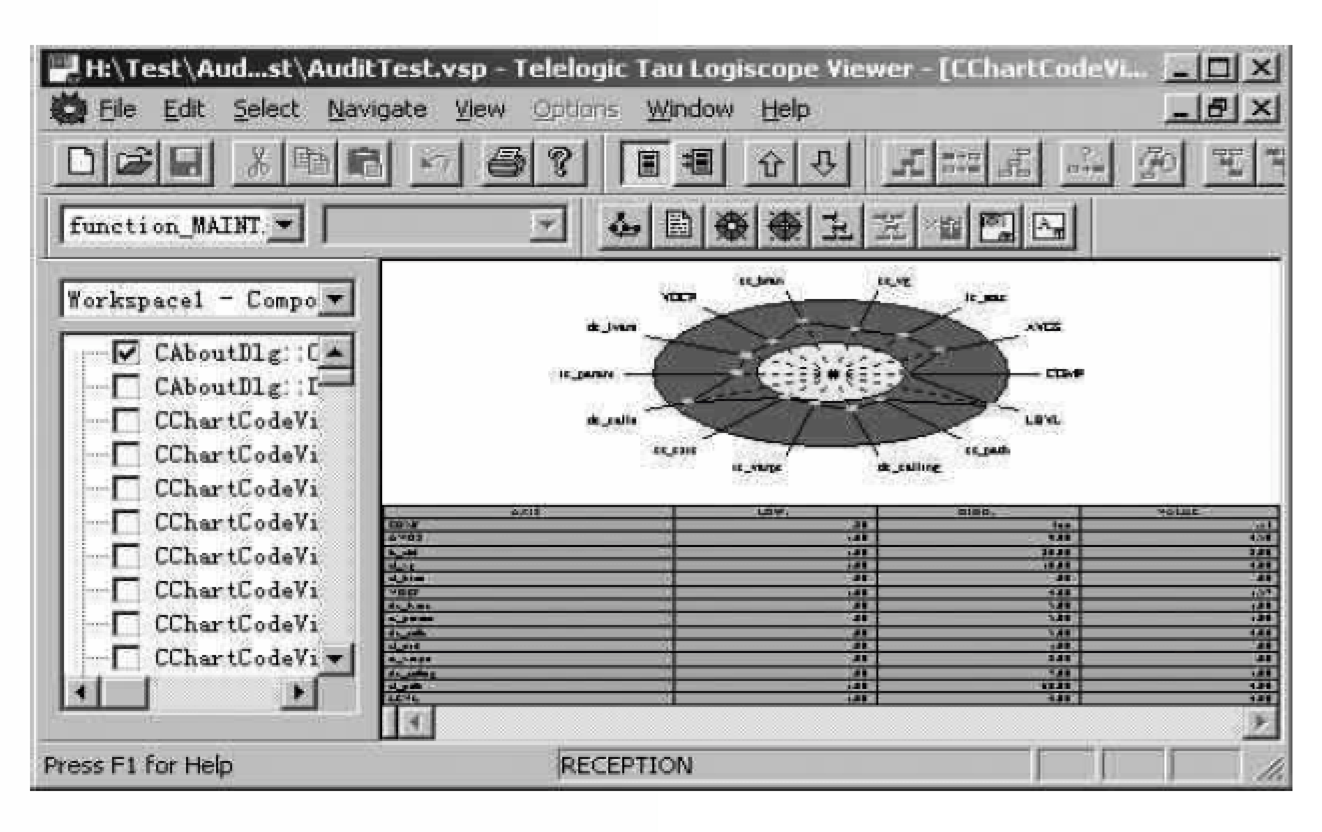


图 16.15　函数度量元

单击函数质量标准按钮，会显示函数质量标准的检测结果如图 16.16 所示。

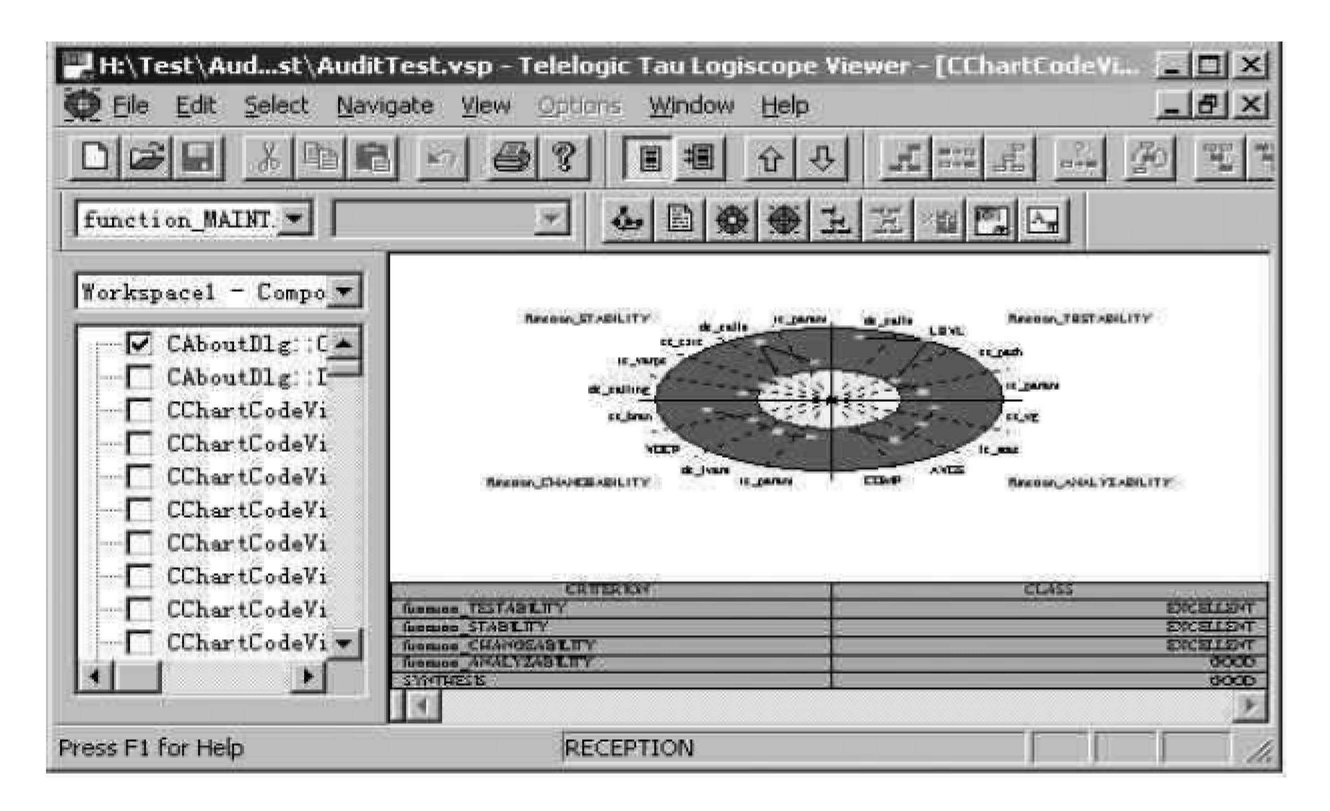


图 16.16　函数质量标准

单击函数调用关系按钮，会显示函数之间的调用关系的检测结果，如图 16.17 所示。

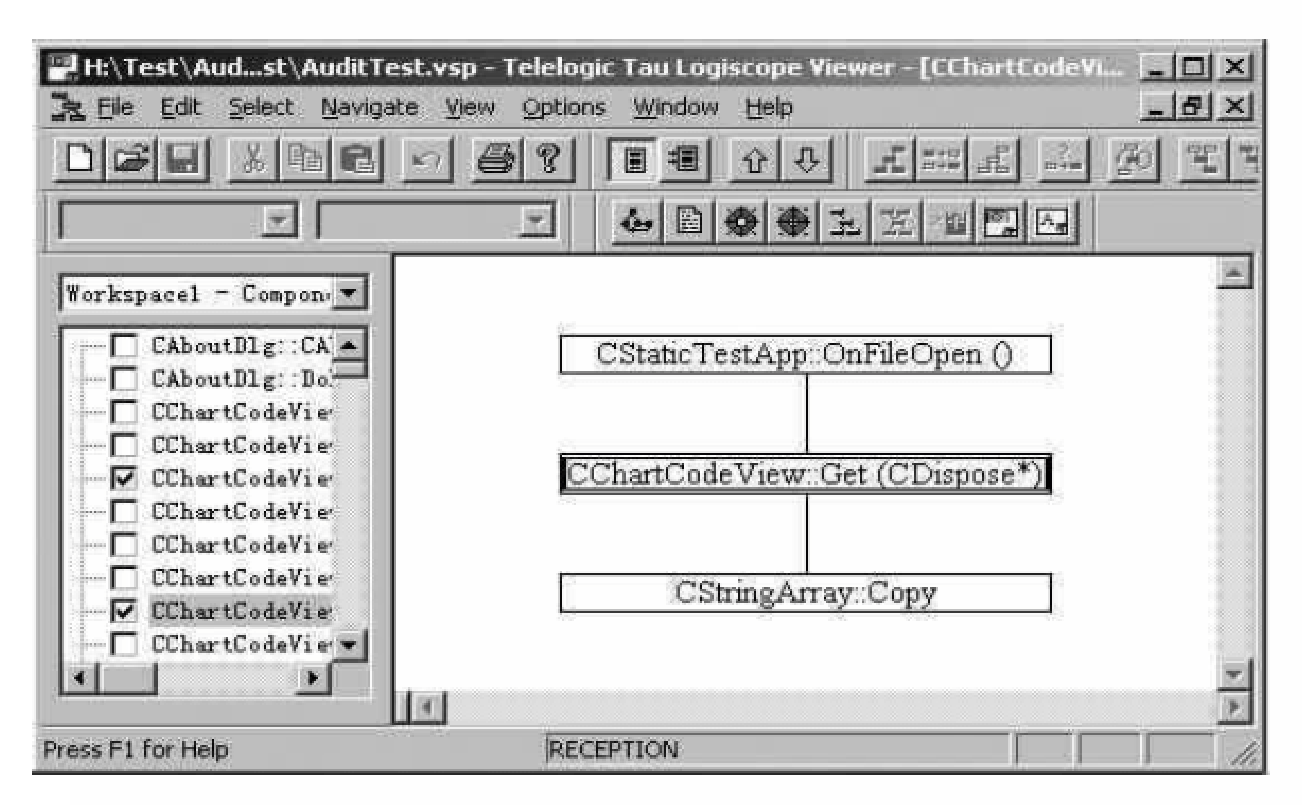


图 16.17　函数调用关系

单击系统度量元按钮，会显示系统度量元的检测结果，如图 16.18 所示。

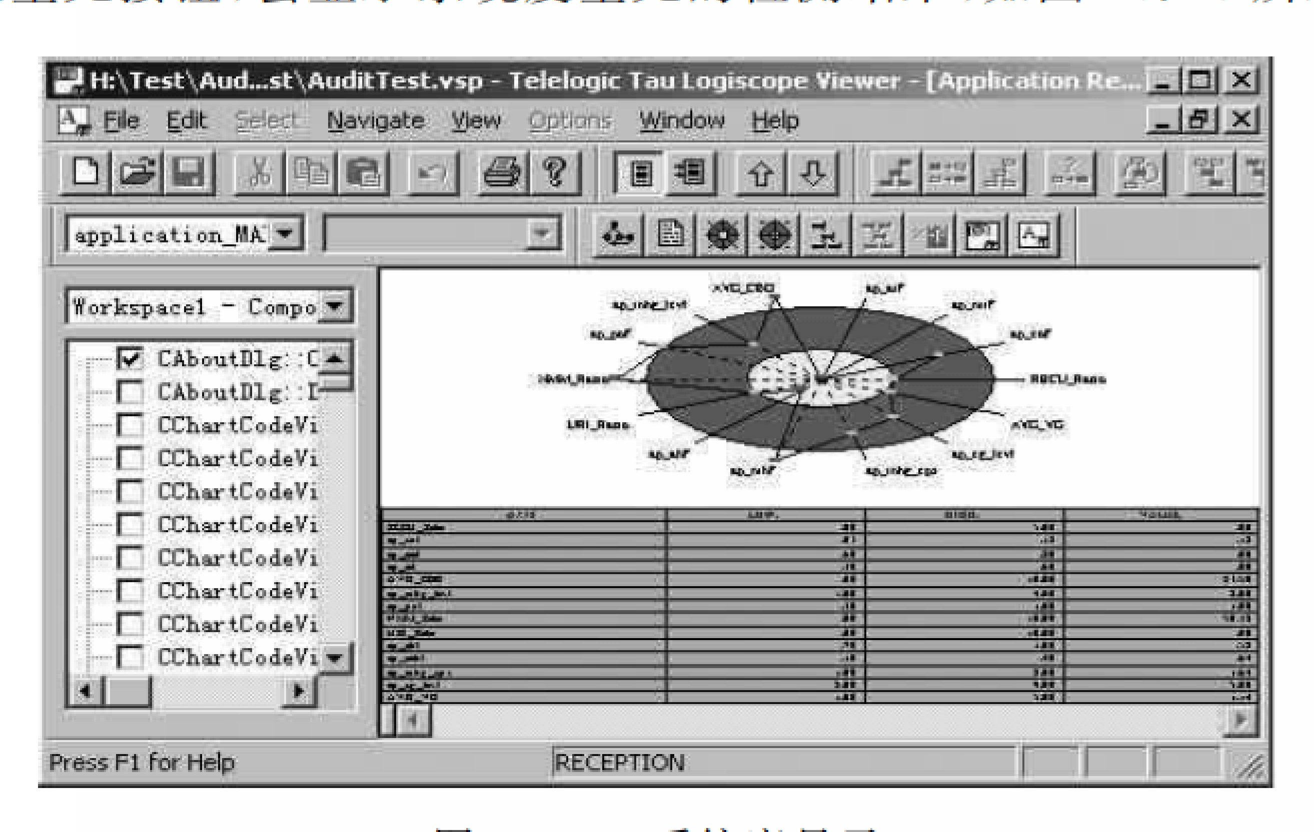


图 16.18　系统度量元

以上显示了函数域、系统域的情况，还可以查看各个类的情况。在 Viewer 中选择 File→New 命令，在弹出的对话框中选中 Class Workspace，如图 16.19 所示，单击“确定”按钮。这时，界面显示如图 16.20 所示。

新建
新建(N)
Function Workspace
Class Workspace
确定
取消
帮助(H)

图 16.19　新建 Class Workspace

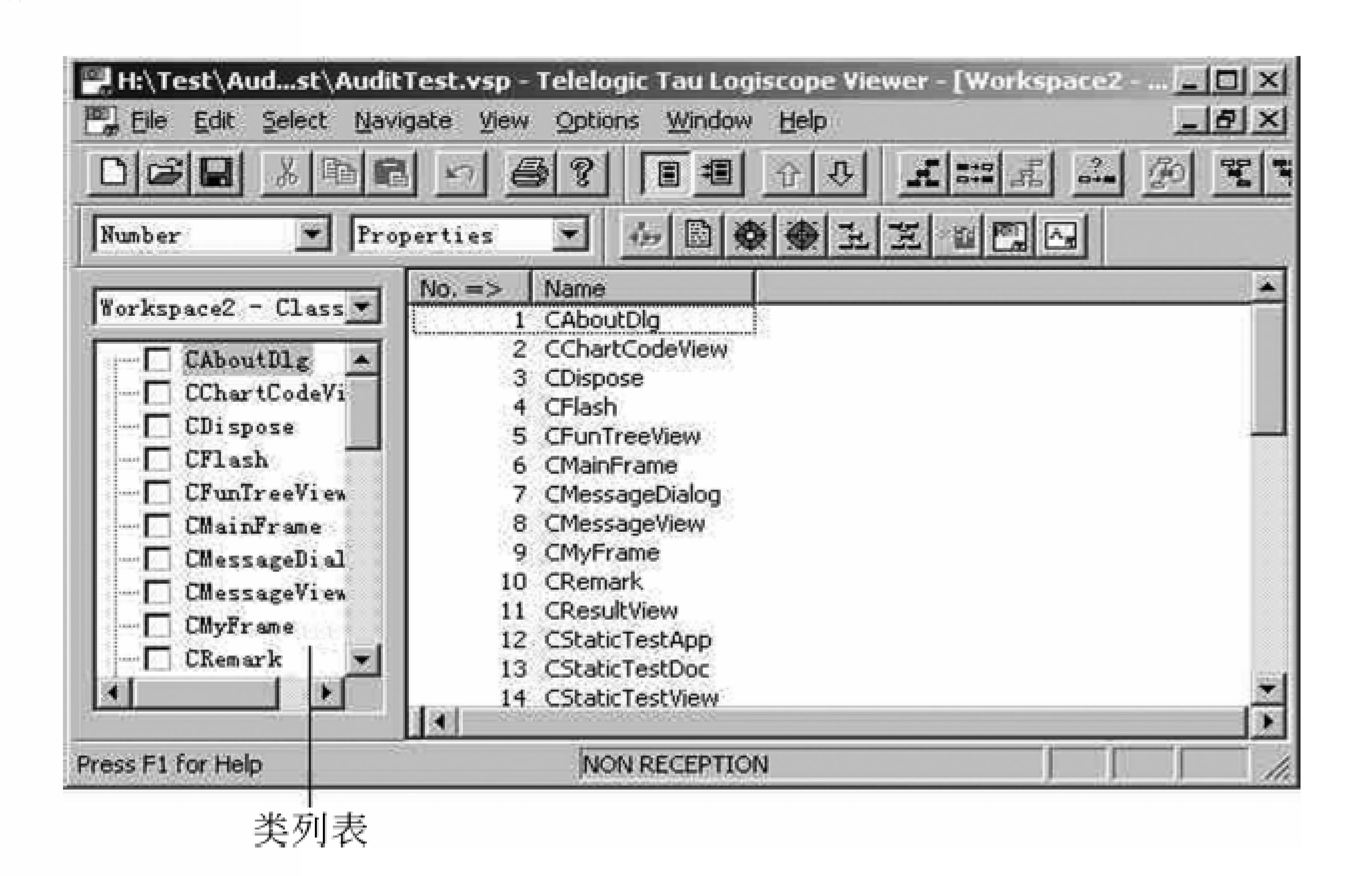


图 16.20　Class Workspace

窗口的列表框中列出了系统中所有的类。选中某个类后，单击下面这个工具条上的按钮，可以查看关于该类的各种分析信息。工具条及各个按钮的功能如图 16.21 所示。

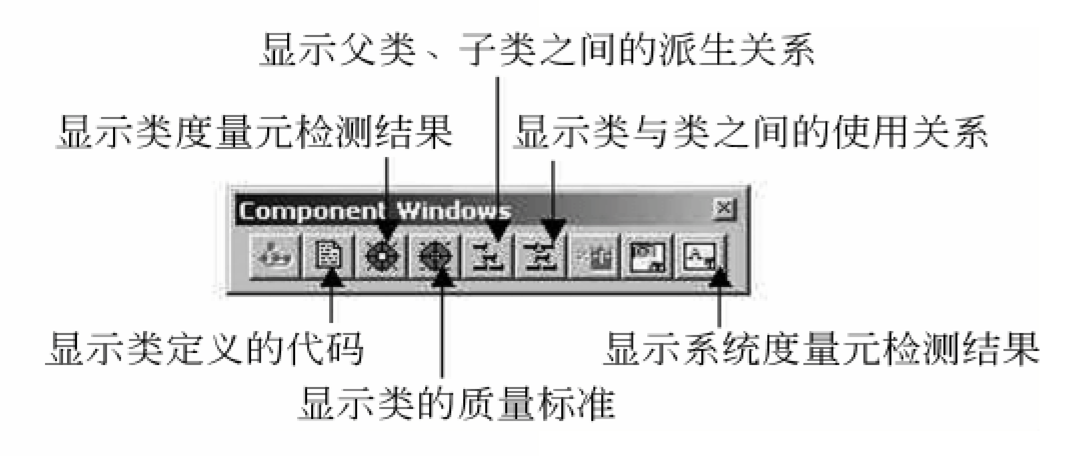


图 16.21　Class Workspace 工具条

单击类度量元按钮，会显示类度量元的检测结果，如图 16.22 所示。

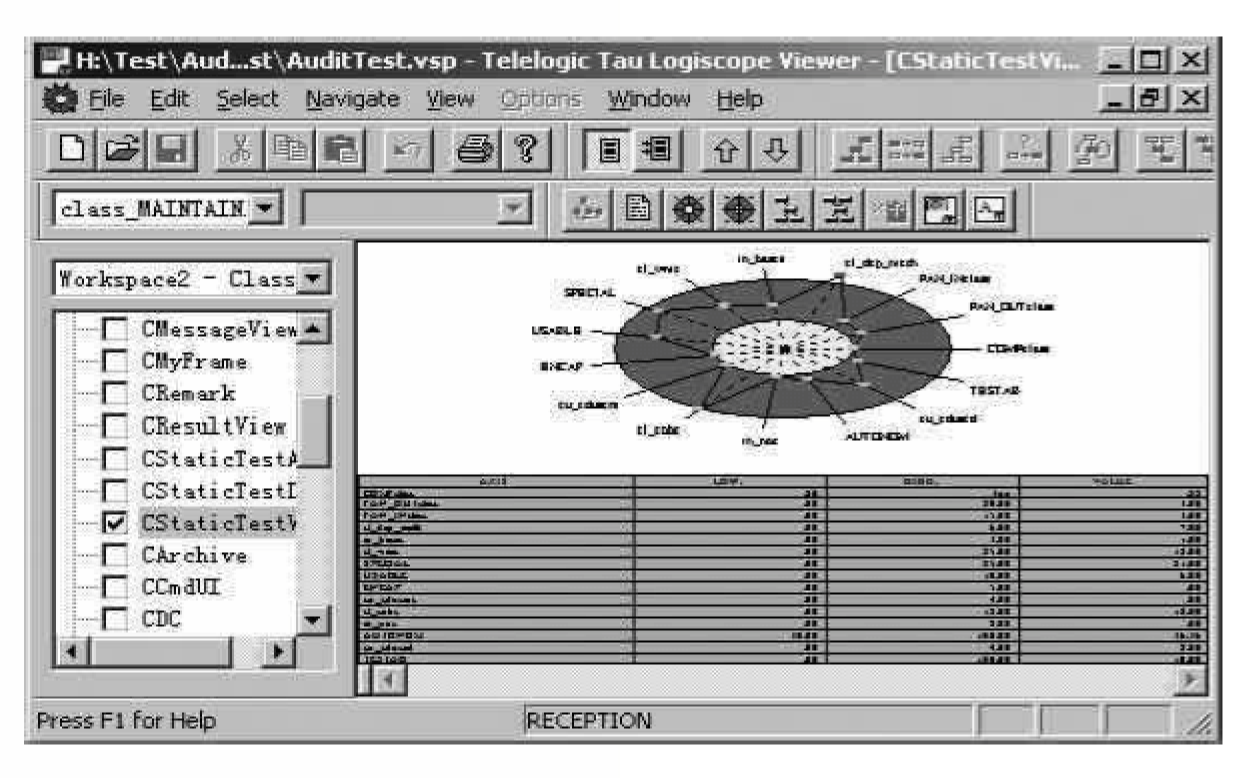


图 16.22　类度量元

单击类质量准则按钮，会显示类质量准则的检测结果，如图 16.23 所示。

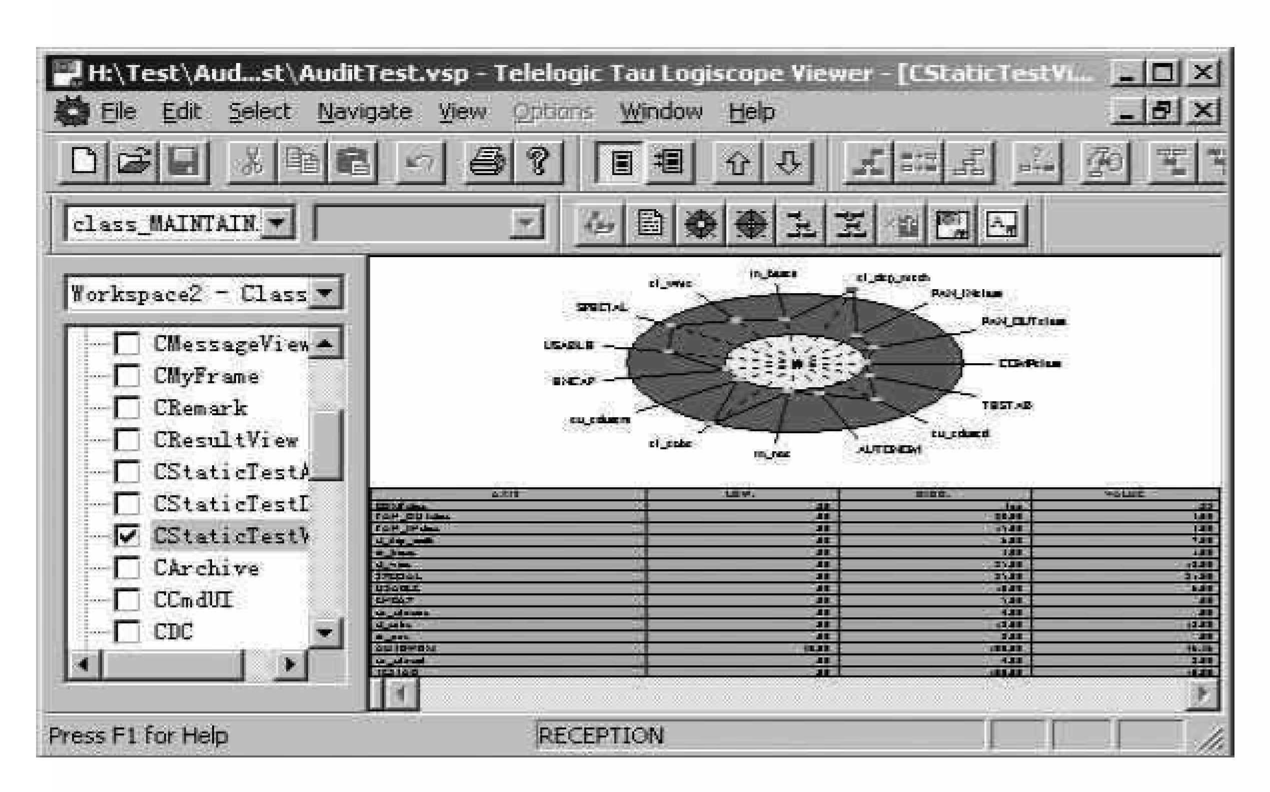


图 16.23　类质量准则

单击类派生关系按钮，会显示父类、子类之间的派生关系，如图 16.24 所示。

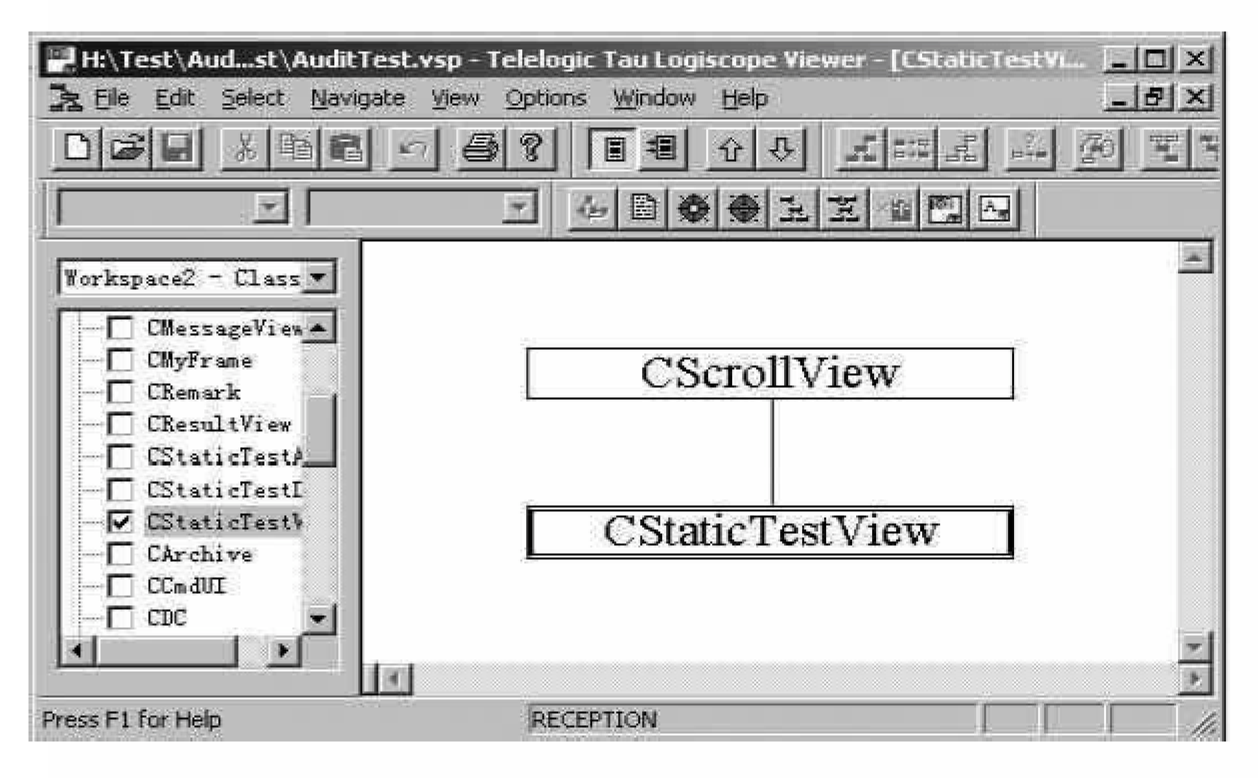


图 16.24　类派生关系

单击类使用关系按钮，会显示类与类之间的使用关系，如图 16.25 所示。

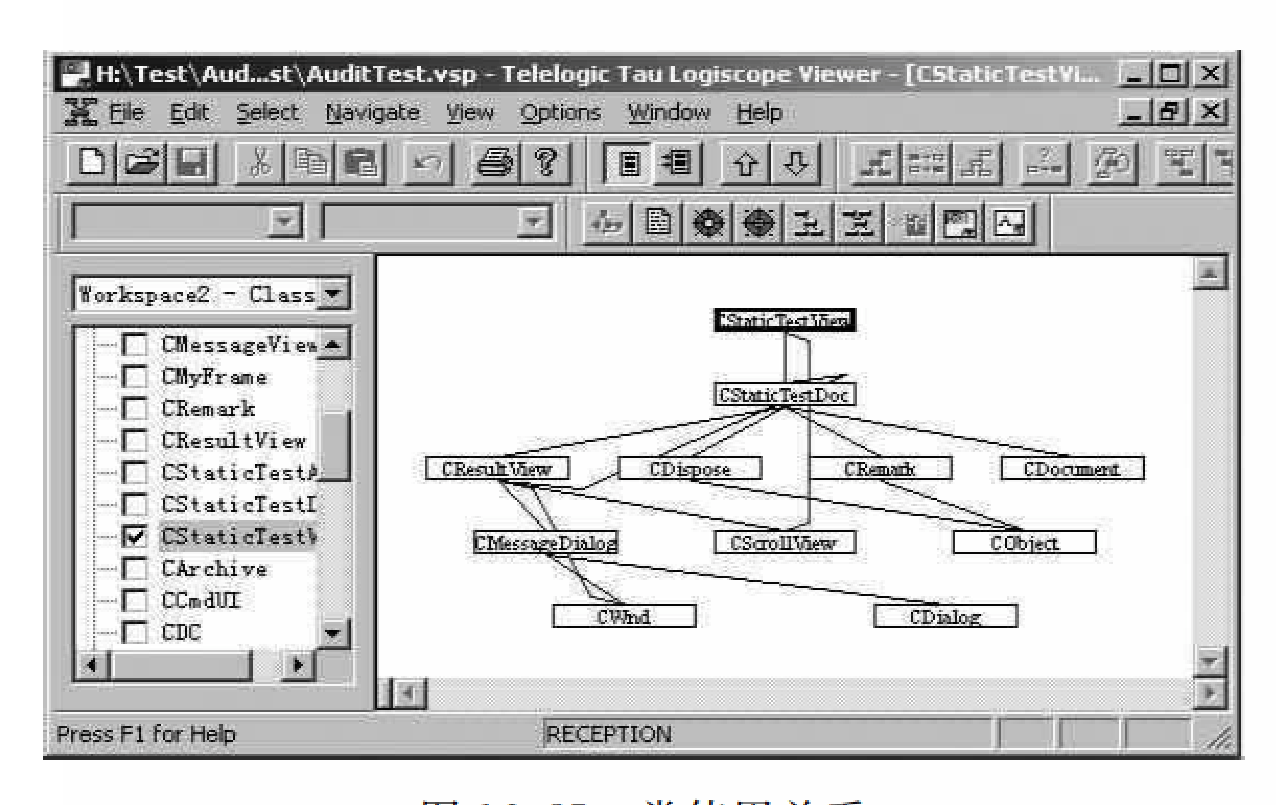


图 16.25　类使用关系

16.2.2　使用 RuleChecker

RuleChecker 是一个静态、白盒性质的测试工具，用来检查代码书写规范性。使用 RuleChecker 来检查代码的规范性分为两个步骤：首先是建立被检测代码的 RuleChecker 项目，然后分析 RuleChecker 给出的代码书写规范性检测结果，得出报告。

下面详细介绍两个步骤。

1. 建立 RuleChecker 项目

1）Logiscope studio 中建立 Audit 项目

（1）选择“开始”→Logiscope studio，进入 Logiscope studio 环境，选择 File→New 命令，弹出如图 16.26 所示对话框。

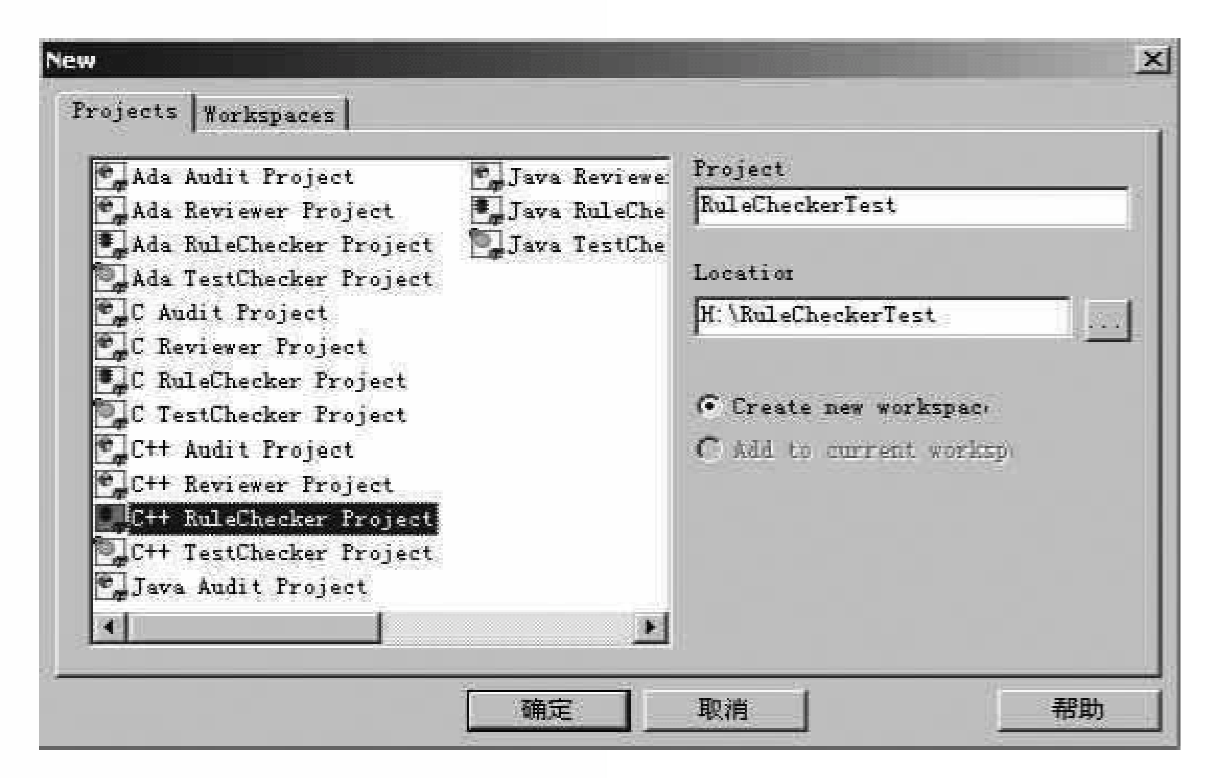


图 16.26　新建 RuleChecker 项目

选中 Project 选项卡后，在列表框中选择 C++ RuleChecker Project 项，然后在 Project 编辑框中添入要建立的这个 RuleChecker 项目的名字，再为 Locatior 编辑框选一个存放生成的 RuleChecker 项目的文件目录。

（2）单击“确定”按钮，弹出如图 16.27 所示对话框。

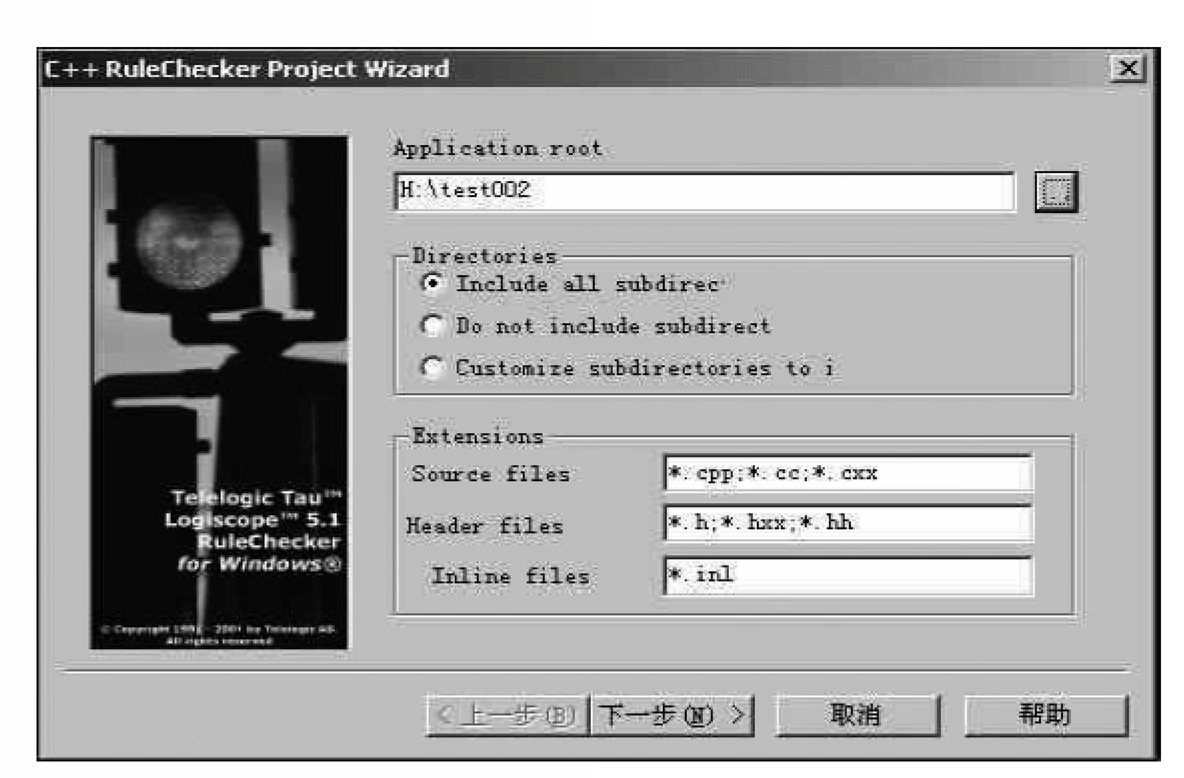


图 16.27　新建 RuleChecker 项目向导

在 Application root 编辑框中添入所要检测的源程序文件的存放路径。

(3) 单击"下一步"按钮,弹出如图 16.28 所示对话框。



图 16.28 新建 RuleChecker 项目向导

使 Choose a parser 组合框保持默认选项,在 Choose a configuration file 编辑框中添入所设计的规则集文件(默认选中 TestChecker 提供的规则集文件,该文件的路径在"LogiscopeHOME\Logiscope\data\audit_c++\RuleChecker.cfg"),其他均采用默认值即可。

(4) 单击"下一步"按钮,弹出如图 16.29 所示对话框。



图 16.29 新建 RuleChecker 项目向导

图中显示将要生成的 RuleChecker 项目的一些情况,单击"完成"按钮,弹出如图 16.30 所示窗口。

选择 Project→Build 命令,RuleChecker 开始扫描程序代码。Build 执行成功之后,检测结果也就产生了。

2) 在 Visual Studio 中建立 RuleChecker 项目

启动 VC6.0,打开要检测的程序(.dsw 文件),选择 Tools→Check Rules 命令,

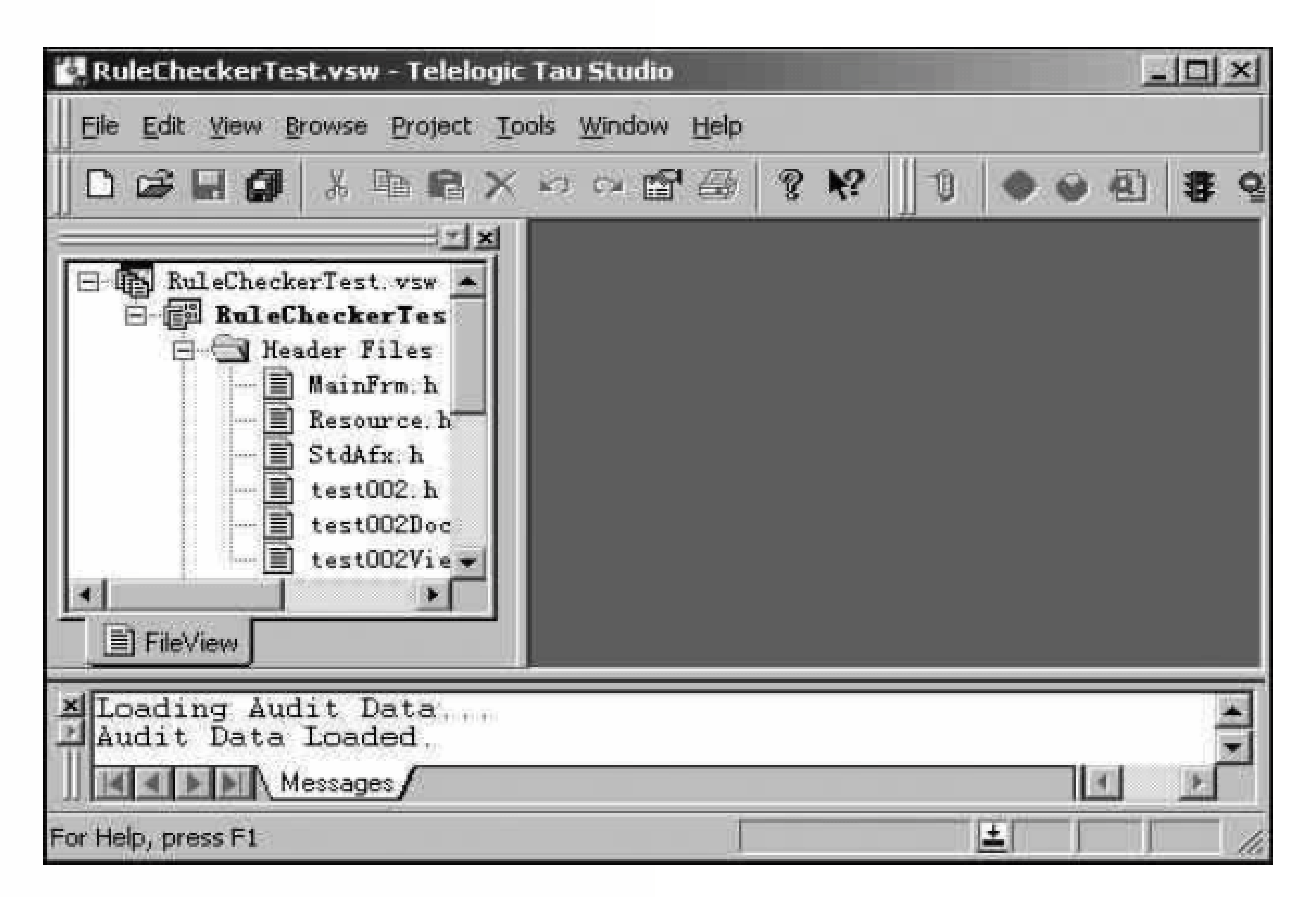


图 16.30　RuleChecker 界面

RuleChecker 开始扫描程序代码，检测代码的书写规范性。操作执行成功之后，选择 Tools→viewer 命令，Logiscope 被启动，显示检测结果。

2. 查看检测结果

选择 Browse→Rule→Rule Violations 命令，RuleChecker 会在树状视图中列出代码中所有违反编码规范的地方，如图 16.31 所示。

在树状视图中共有三个文件夹——Violated Rules 文件夹、Clean Rules 文件夹、Ignored Rules 文件夹。其中，Violated Rules 文件夹罗列出了代码未遵守的编码规范；Clean Rules 文件夹罗列出了代码遵守的编码规范；Ignored Rules 文件夹罗列出了在本次检测中忽略的编码规范。各文件夹展开后，如图 16.32 所示。

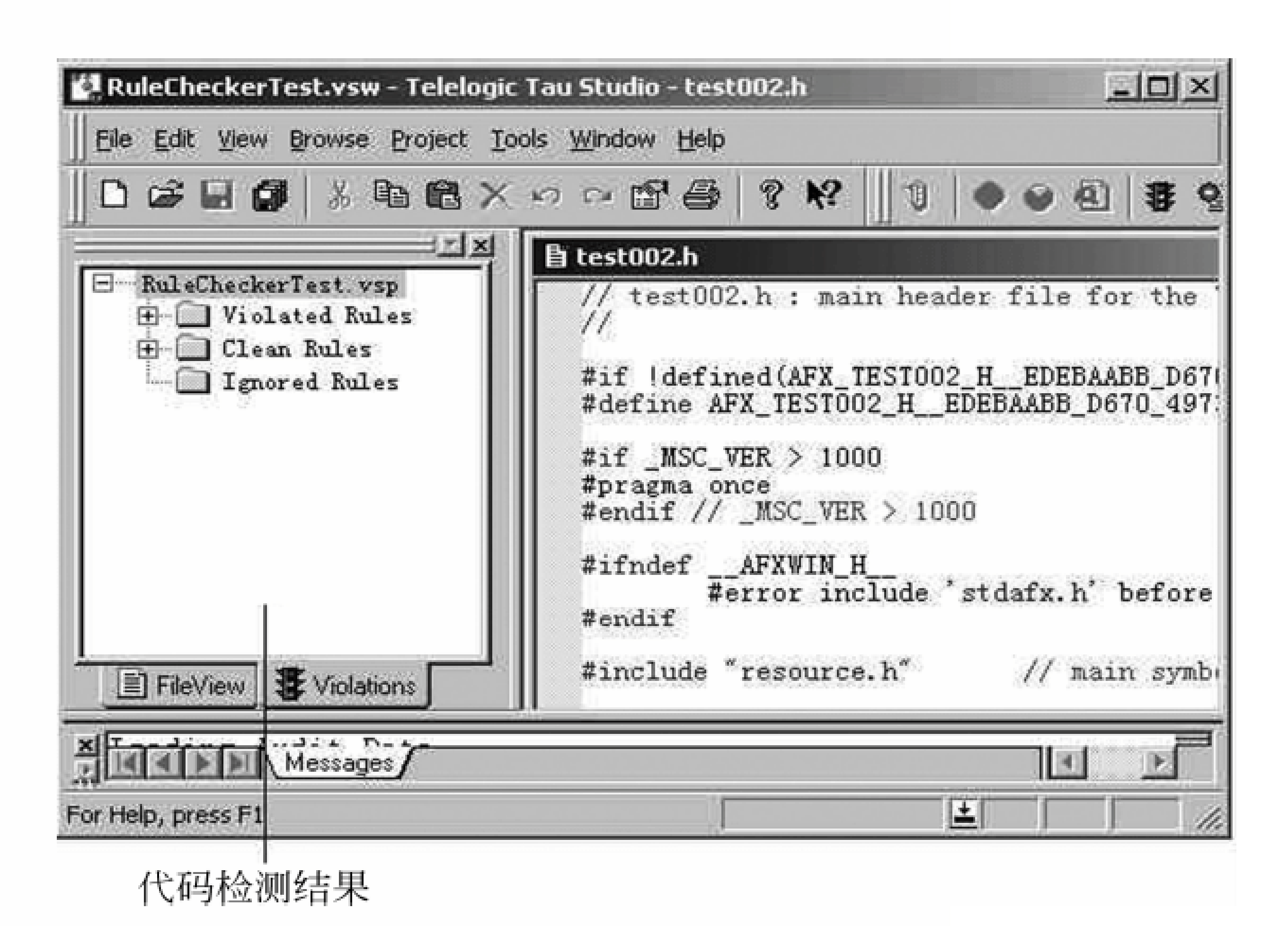


图 16.31　RuleChecker 界面

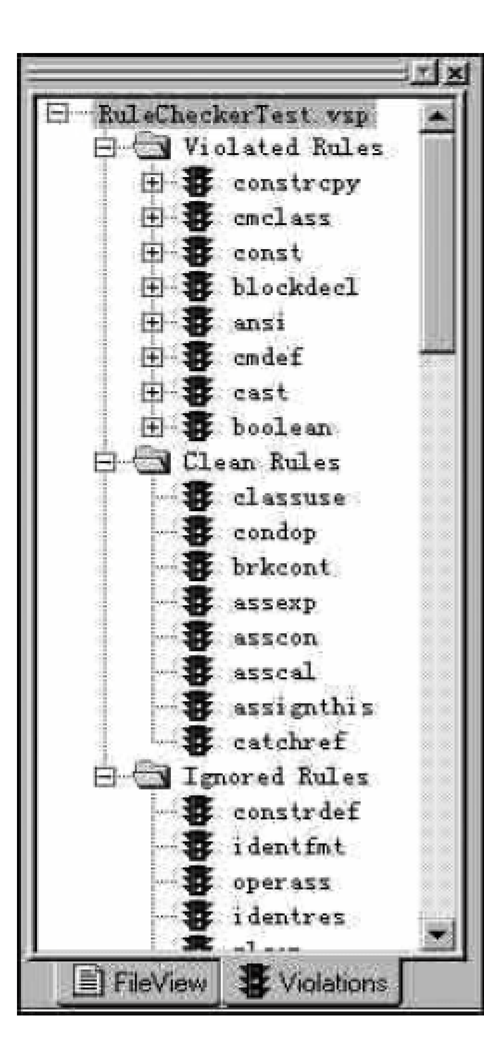


图 16.32　树状视图

展开 Violated Rules 文件夹后，显示了在代码中未遵守的各项编码规范，每个规范都以一个交通灯的图标显示。用双击这个图标，RuleChecker 会显示对这条编码规范的解释，如图 16.33 所示。

单击图标左侧“⊞”将其展开，列出违反该项编码规范的源文件的文件名，显示在该文件中违反该编码规范的代码的行号，如图 16.34 所示。

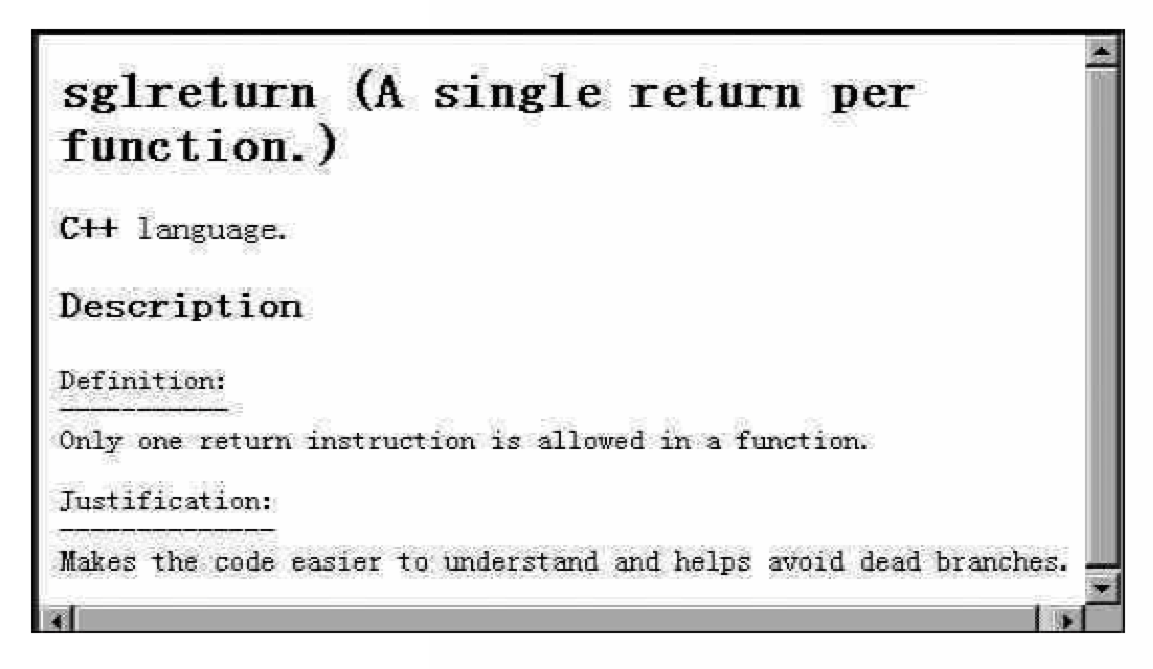


图 16.33　编码规范解释

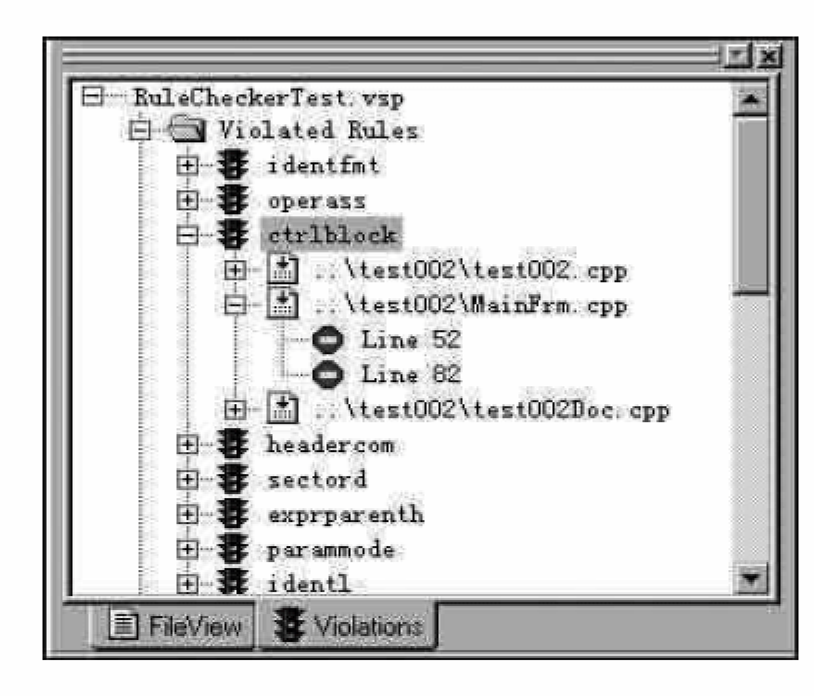


图 16.34　违反编码规范的位置

双击这个行号，RuleChecker 会显示源文件，并将光标定位到违反该规范的代码行处，如图 16.35 所示。

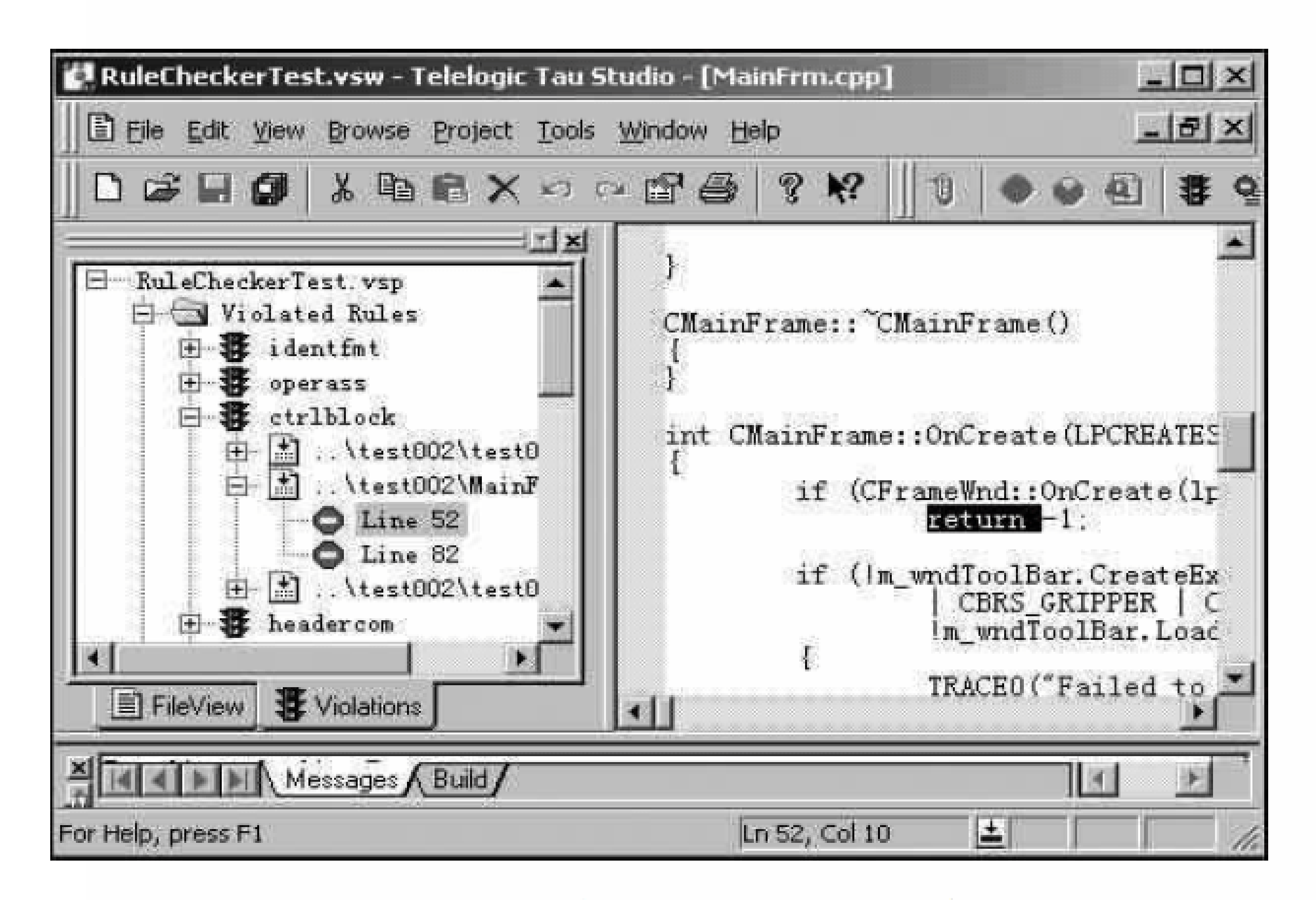


图 16.35　定位违反编码规范的代码

单击 Project→Settings 命令，启动如图 16.36 的一个对话框，开启、关闭某些编码规范。

最下面的那个列表框列出了 RuleChecker 提供的所有编码规范，当其前面的复选框是选中状态时，则该规范在 RuleChecker 检测过程中生效，当其前面的复选框是未选中状态时，则该规范在 RuleChecker 检测过程中不生效。根据具体情况，使某些前面生效，或不生效。设置完成后，单击“确定”按钮，保存设置。然后选择 Project→Build 命令，重新让 RuleChecker 扫描代码。Build 结束后，产生与设置相符的检测结果。

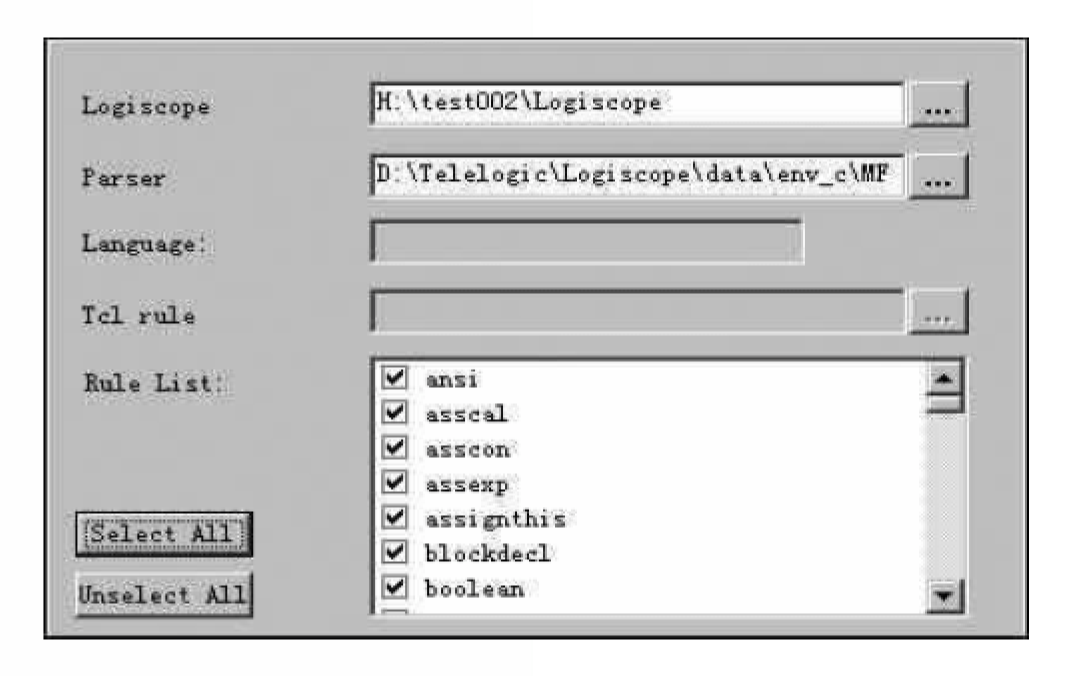


图 16.36　开启、关闭某些编码规范

选择 Browse→Rule→Rule Violations Report 命令，会生成 RuleChecker 的检测报告，如图 16.37 所示。



图 16.37　检测报告

报告分两部分：第一部分分别以源文件为单位和以编码规范为单位，将检测结果以表格的形式显示了出来。第二部分，给出了所有编码规范的解释说明。

16.2.3　使用 TestChecker

TestChecker 是白盒、动态测试工具，用于统计被测试程序的测试覆盖率。TestChecker 重点统计的覆盖率是边覆盖率，也叫判定到判定的覆盖。

TestChecker 统计被测试程序的测试覆盖率分为两个步骤：首先是建立被测程序的 TestChecker 项目；然后，在 TestChecker 环境中运行被测程序，执行测试用例，TestChecker 会给出执行测试用例后的覆盖率。

下面对这两个步骤分别进行介绍。

1. 建立 TestChecker 项目

1）在 VC6.0 中进行设置

（1）选择“开始”→Logiscope studio 命令，进入 Logiscope Studio 环境，选择 File→New 命令。用 VC 6.0 打开要测试项目(.dsp 或.dsw 文件)。

（2）VC 6.0 启动后，选择 Build→Configurations 命令。

（3）单击 Add 按钮，添加一个名为 Logiscope 的文件夹，如图 16.38 所示。

（4）单击 OK 按钮，打开的配置文件夹对话框如图 16.39 所示。

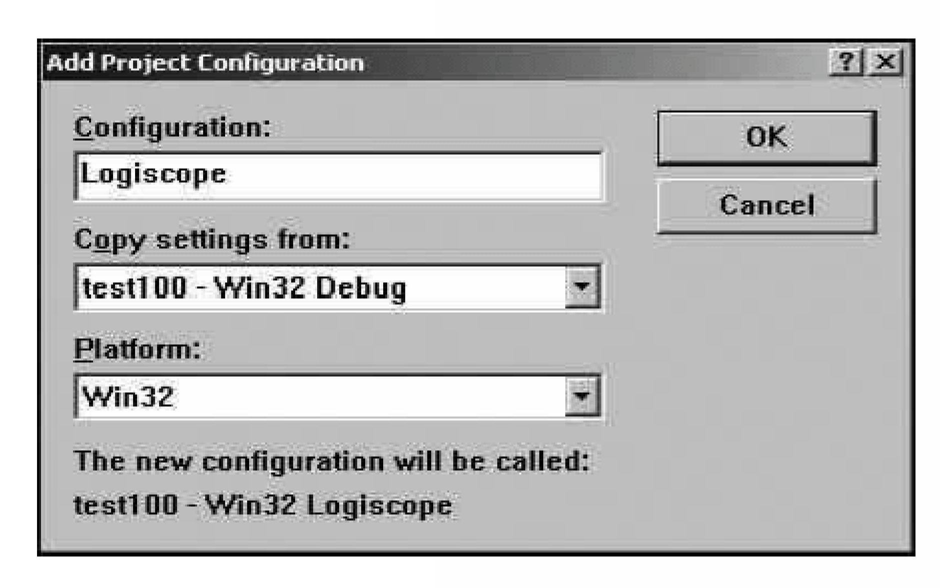



图 16.38　添加文件夹对话框

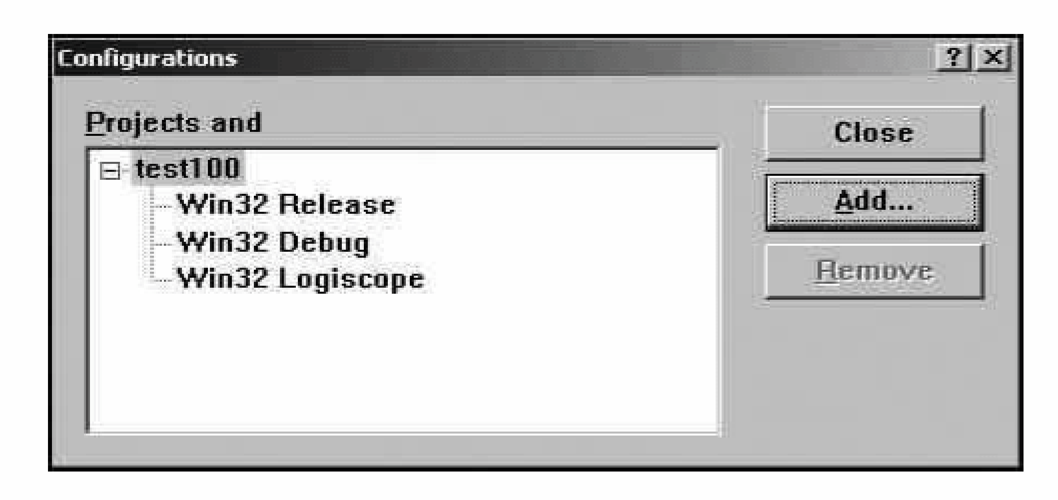


图 16.39　配置文件夹对话框

（5）选择 Build→Set Active Configuration 命令，选中 Logiscope 选项，如图 16.40 所示。

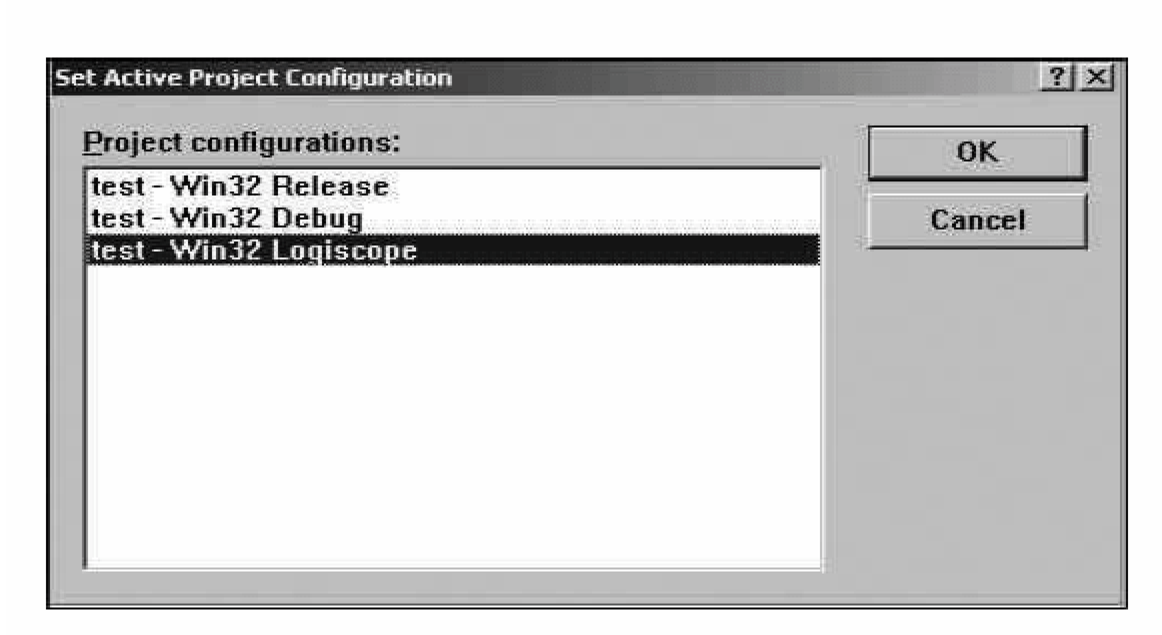


图 16.40　设置当前文件夹对话框

（6）选择 Project→Settings 命令，在 VC 6.0 的 Settings 中进行一些设置。

（7）设置 C/C++ 选项卡。

选中 C/C++ 选项卡，在 Category 组合框中选中 Preprocessor，在 Additional include directories 编辑框中添入计算机上 Logiscope 的 Include 文件夹的路径，该文件夹的路径为“Logiscope 安装目录\Logiscope\instr\include”，如图 16.41 所示。

（8）设置 LINK 选项卡。

切换到 LINK 选项卡，在 Object/library modules 编辑框中输入 vlgtc.lib，如图 16.42 所示。

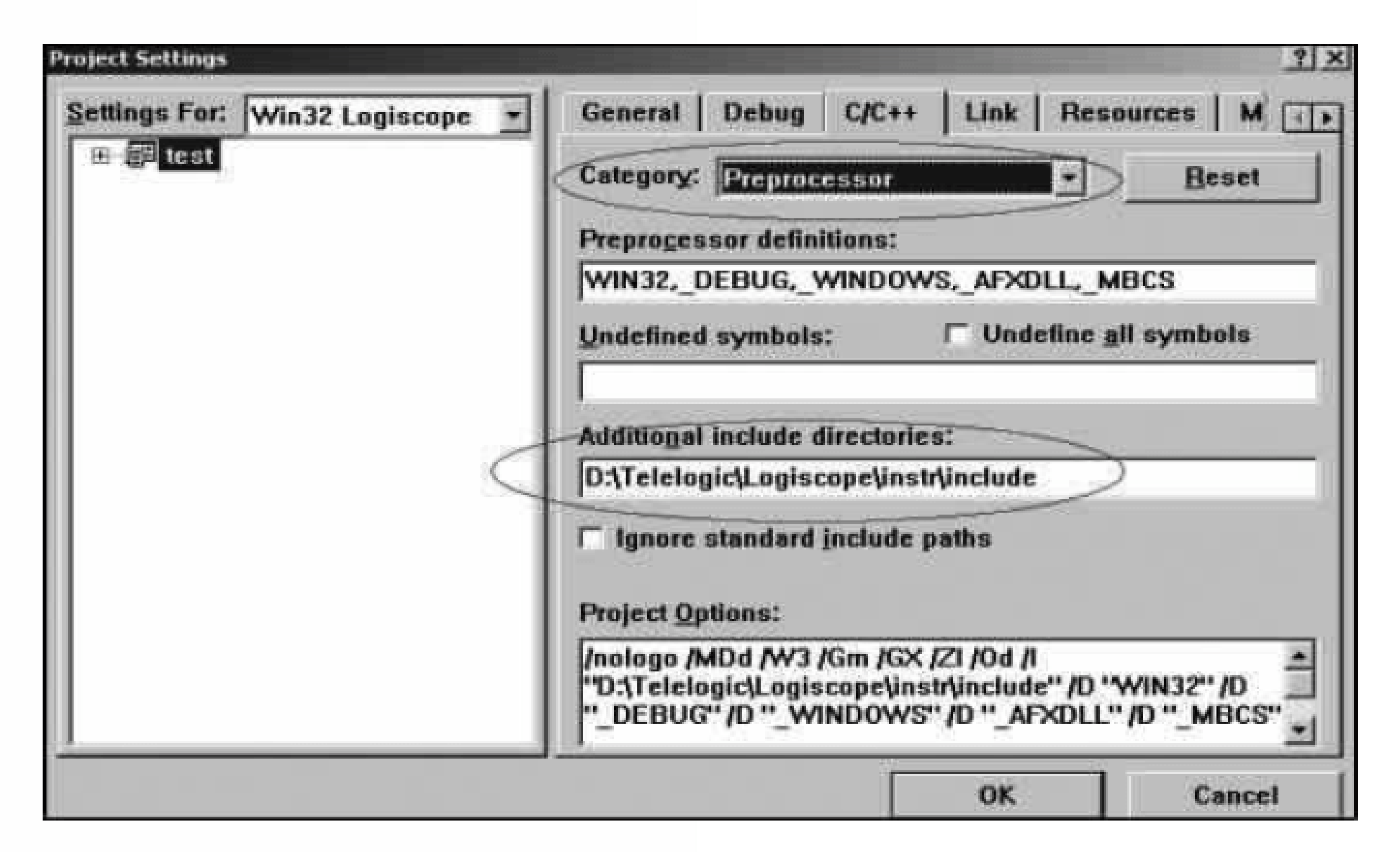


图 16.41　Project Settings 对话框 1

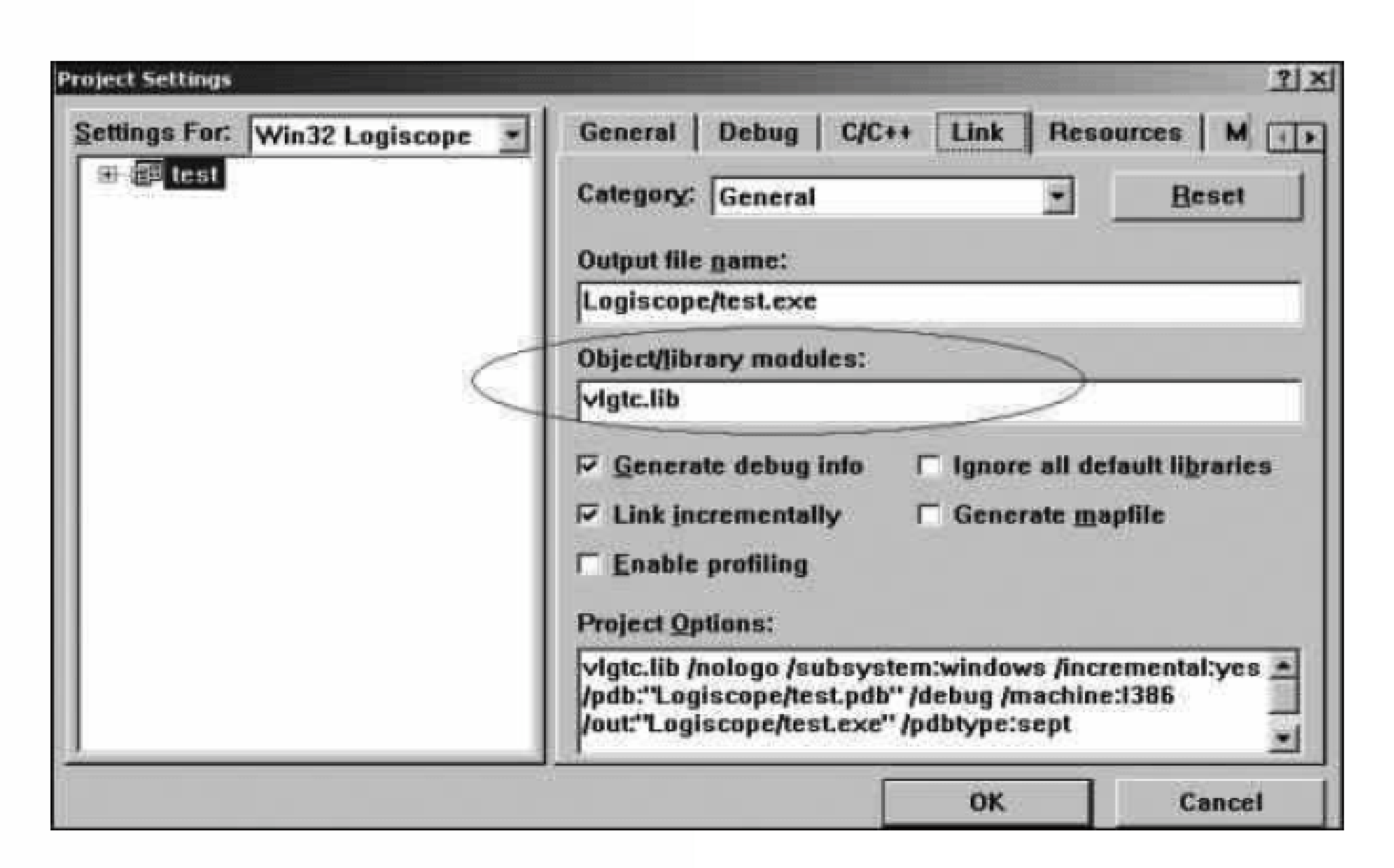


图 16.42　Project Settings 对话框 2

然后，在 Category 组合框中选中 Input，在 Additional library path 编辑框中为上面这个 lib 文件指定路径，路径为“Logiscope 安装目录\Logiscope\instr\lib”。设置情况如图 16.43 所示。

(9) 选择 Project→Export Makefile 命令，打开如图 16.44 所示对话框。

(10) 选择 File→Save All 命令，保存所做的一切设置。

至此，在 VC6.0 中对被测程序的设置全部完成了。退出 VC6.0，启动 Logiscope Studio，进入 Logiscope Studio 环境，开始插装被测程序。

2) 在 Logiscope Studio 中插装被测程序

(1) 启动 Logiscope Studio 后，选择 File→New 命令，弹出如图 16.45 所示的对话框。

在 Projects 列表框中选中 C++ TestChecker Project，在 Project 编辑框中为 TestChecker 项目取一个名字。在 Location 编辑框中，建立一个存放路径。

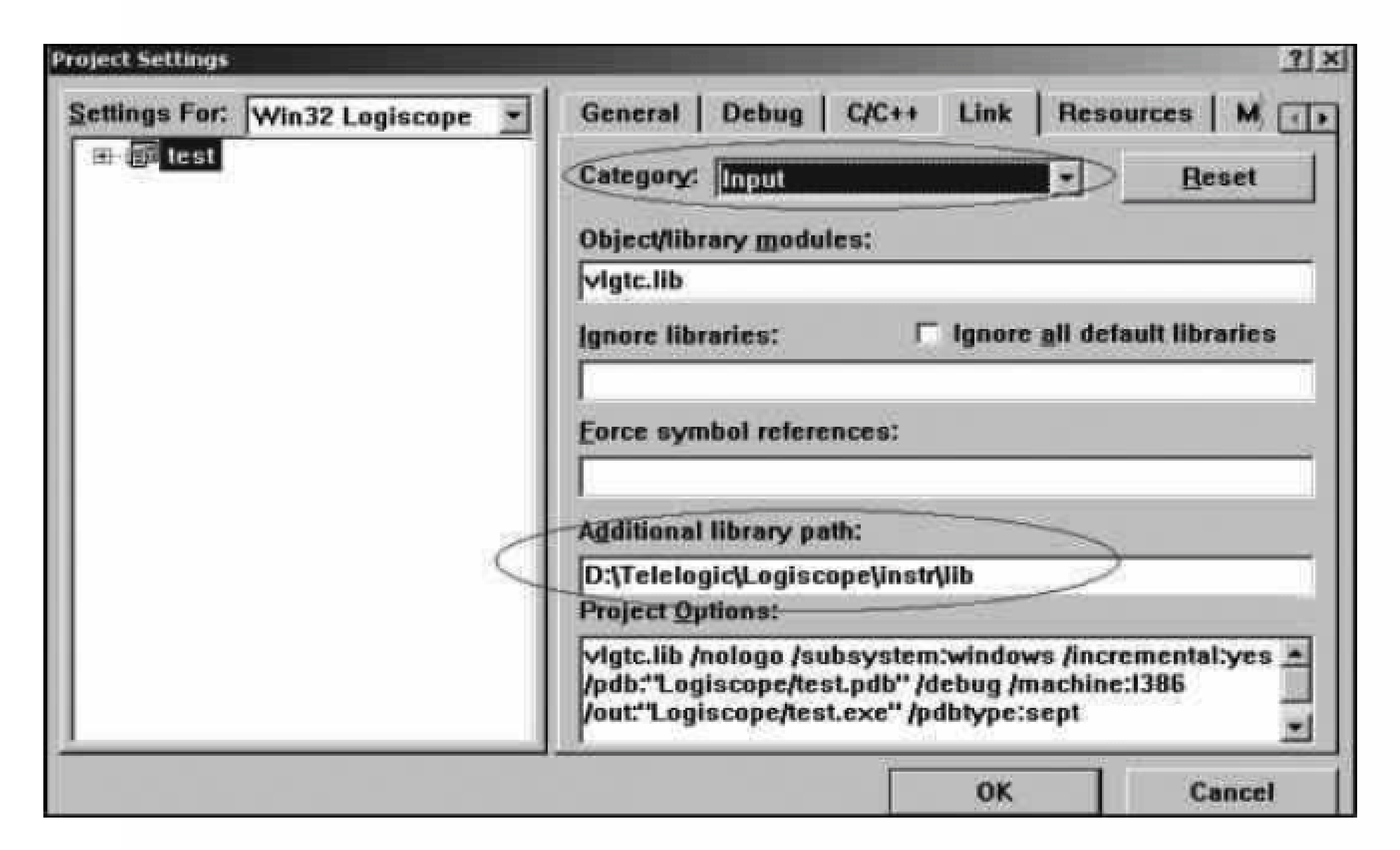


图 16.43　Project Settings 对话框 3

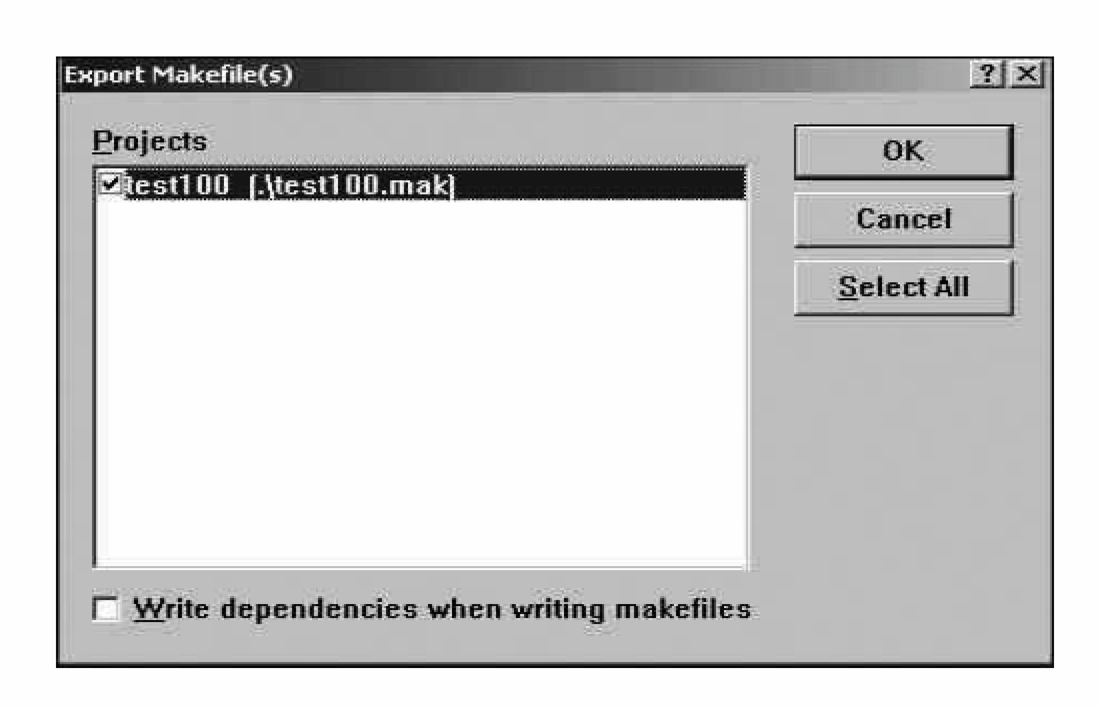


图 16.44　生成.mak 文件对话框



图 16.45　新建 TestChecker 项目对话框

(2) 单击“下一步”按钮,弹出如图 16.46 所示的对话框。

在 Application root 编辑框中,指出所要测试的项目(.dsw 文件)的路径。

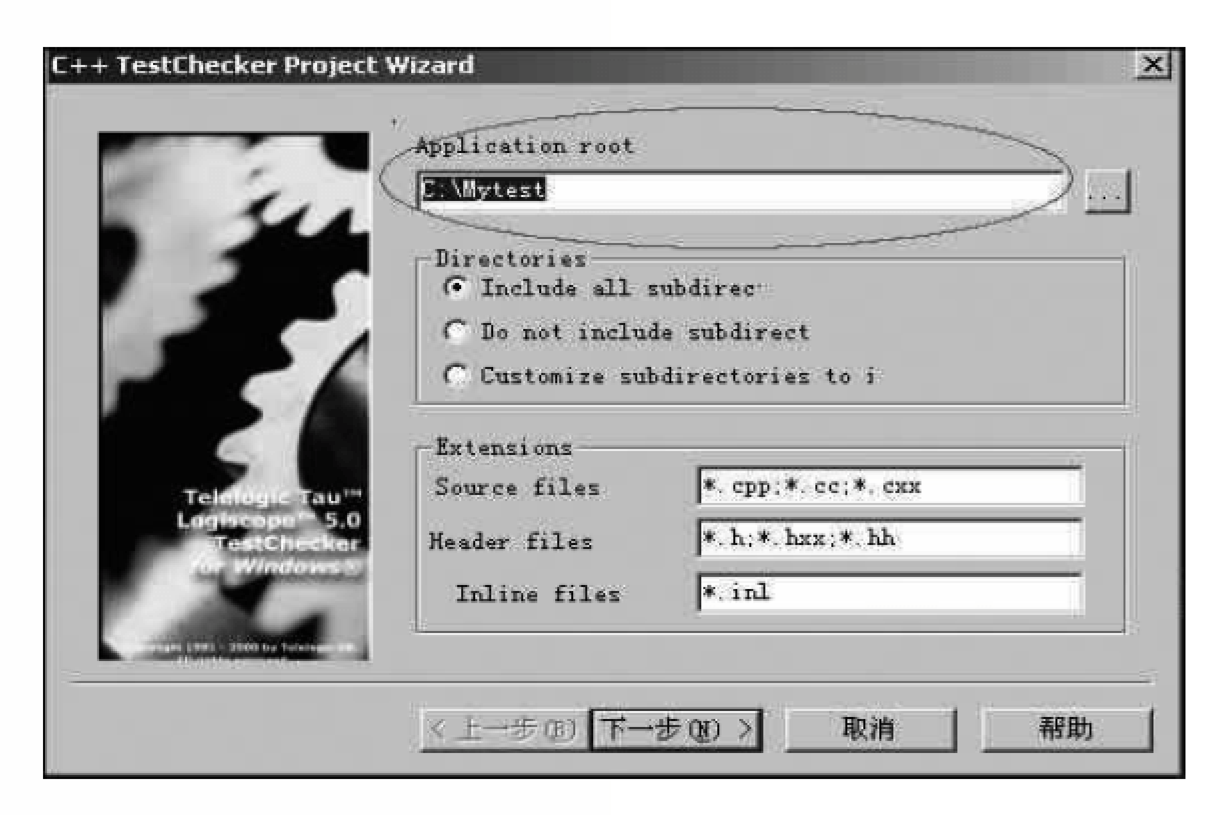


图 16.46　新建 TestChecker 项目向导 1

(3) 单击"下一步"按钮,弹出如图 16.47 所示的对话框。

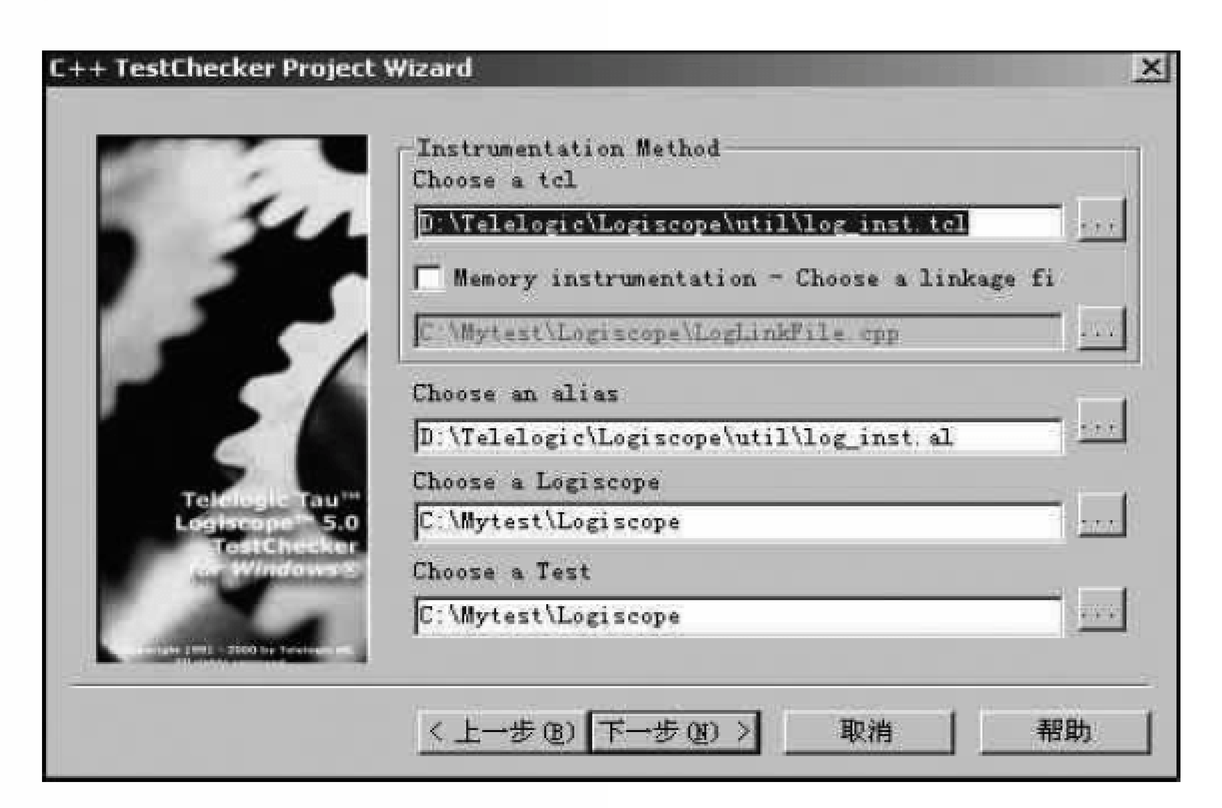


图 16.47　新建 TestChecker 项目向导 2

(4) 单击"下一步"按钮,弹出图 16.48 的对话框。

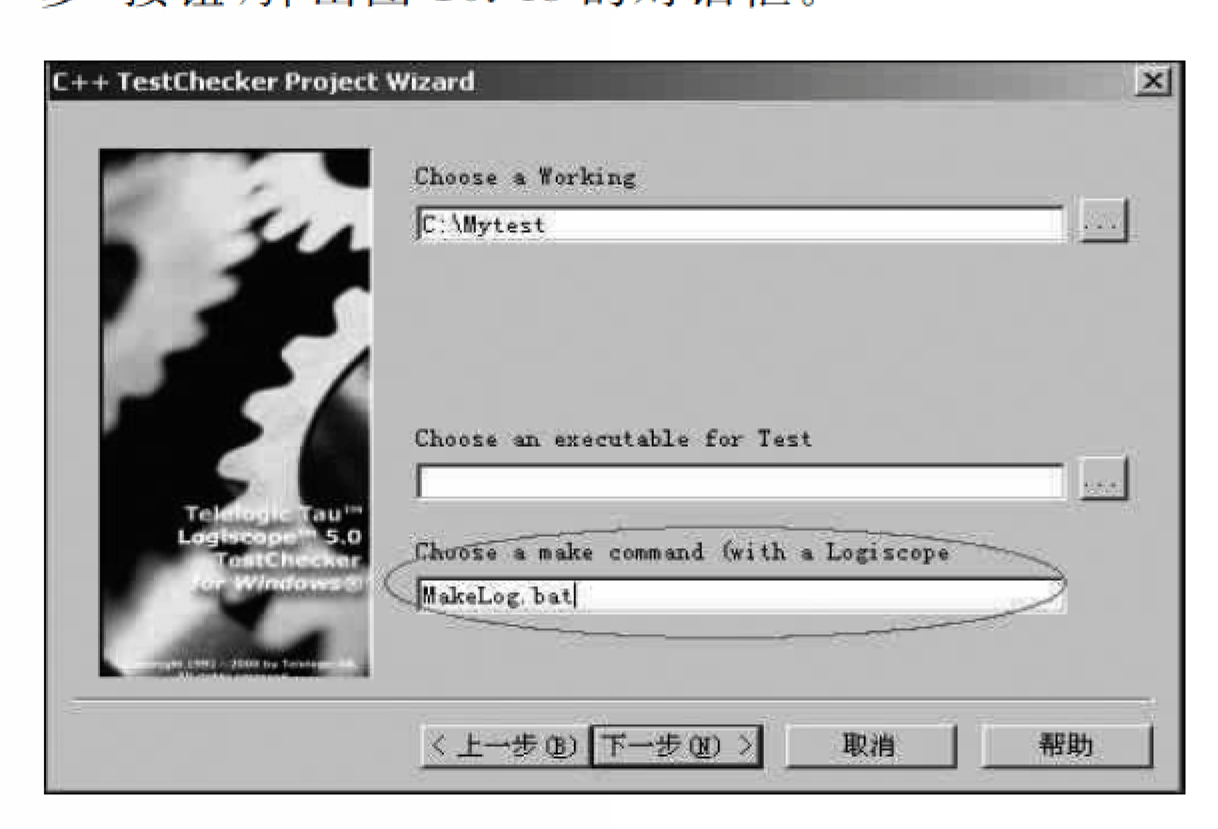


图 16.48　新建 TestChecker 项目向导 3

在 Choose a make command 编辑框中写入 makelog. bat。

(5) 单击"下一步"按钮,弹出图 16.49 的对话框。



图 16.49 新建 TestChecker 项目向导 4

单击"完成"按钮,弹出如图 16.50 所示窗口。



图 16.50 建立 TestChecker 项目

(6) 删除 resource. h 文件。

在 FileView 中,选中 Resource. h 文件,将其删除,如图 16.51 所示。

(7) 编写 makeLog. bat 文件。

在与被测试项目的. dsw 文件同一目录下,新建一个文本文件,其如下的内容:

```
call C:\program files\microsoft visual studio\vc98\bin\vcvars32.bat
nmake /A /F ABCD.mak CFG=ABCD-Win32 Logiscope
```

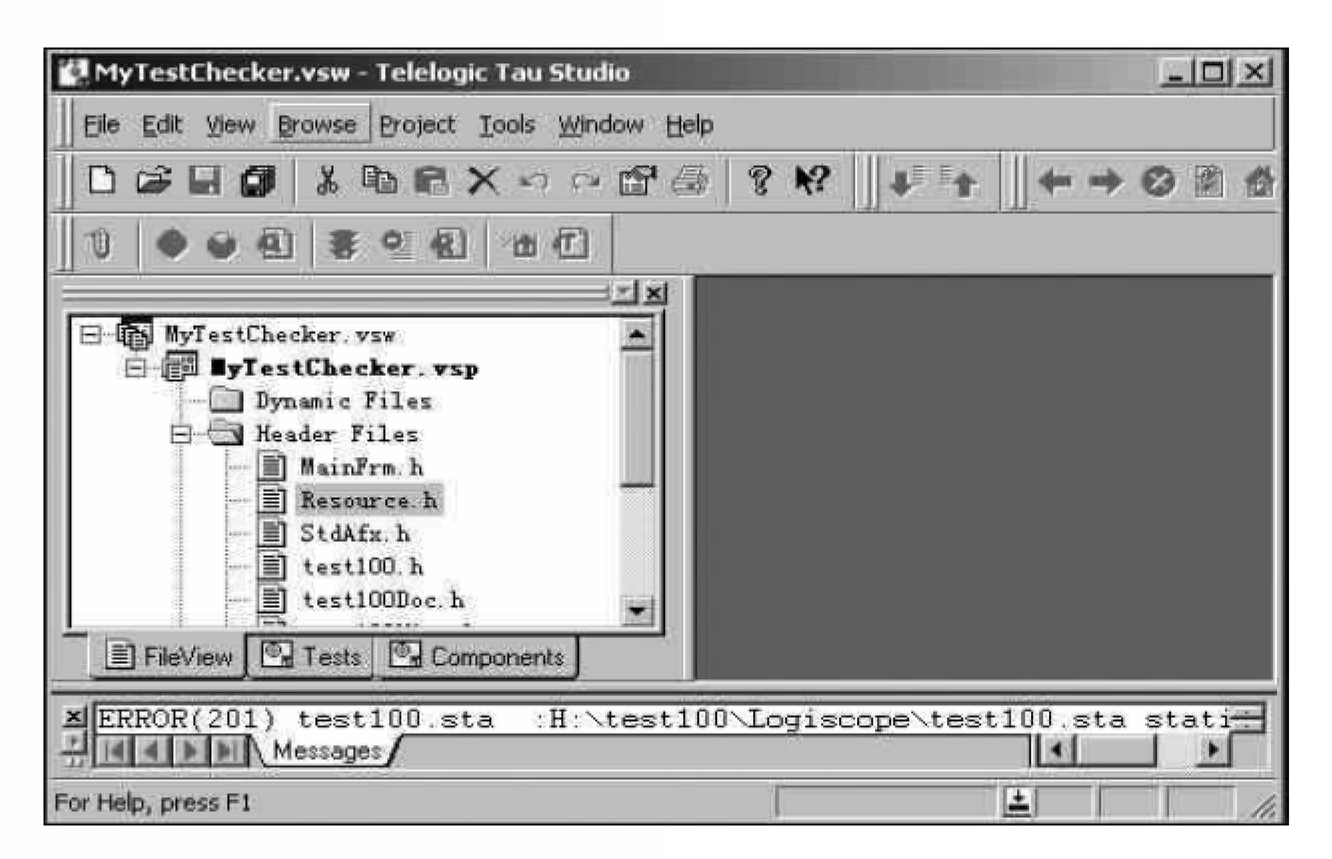


图 16.51　删除 resource.h 文件

其中：

- 第一行的“C:\program files\microsoft visual studio\vc98\bin\vcvars32.bat”，指定的是 VC6.0 安装目录下的 vcvars32.bat 文件的路径。
- 第二行的 ABCD，要替换为所测的项目的名字。

在确保该文件的内容正确后，保存文件，并将文件重命名为 makeLog.bat。

(8) 选择 Project→Build 命令，TestChecker 开始编译，生成可执行程序。

(9) 执行了上一步的操作后，会在所测项目的 Logiscope 文件夹下生成一个 EXE 文件。选择 Project→Settings 命令，在弹出的对话框中选中 TestChecker 选项卡，如图 16.52 所示。

在 Executable for Test 编辑框中选中 TestChecker 生成的.exe 文件。最后的设置结果如图 16.53 所示。

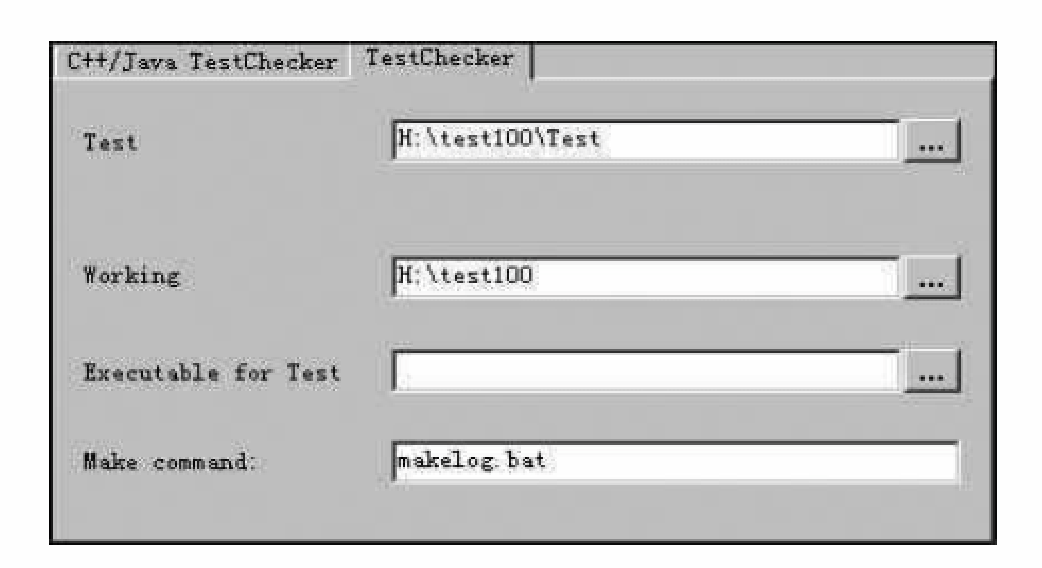


图 16.52　TestChecker 选项卡

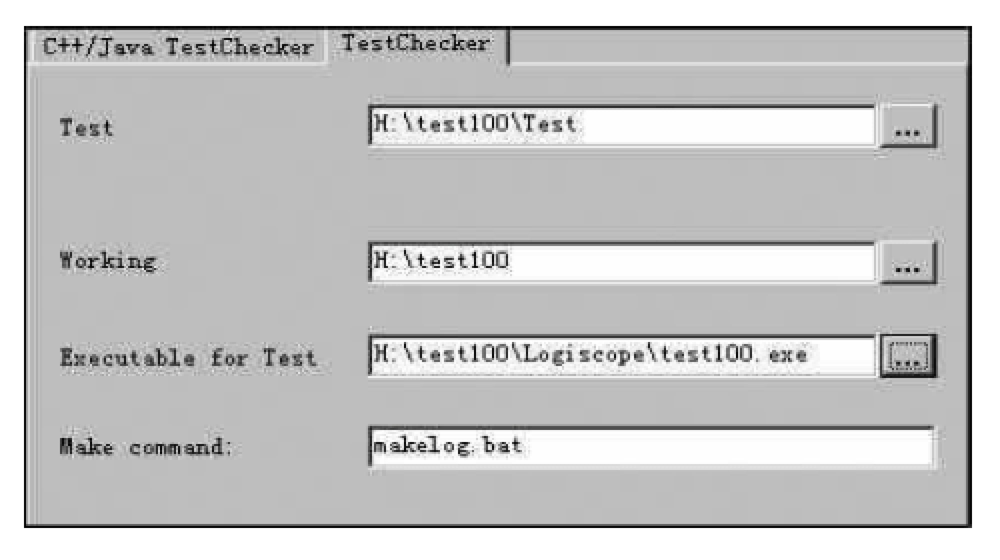


图 16.53　选中可执行文件

(10) 到此为止，一个 TestChecker 项目就全部建立完成了。

2. 用 TestChecker 统计覆盖率

在 Logiscope Studio 中选择 Project→Start TestChecker 命令，启动 TestChecker，如图 16.54 所示。

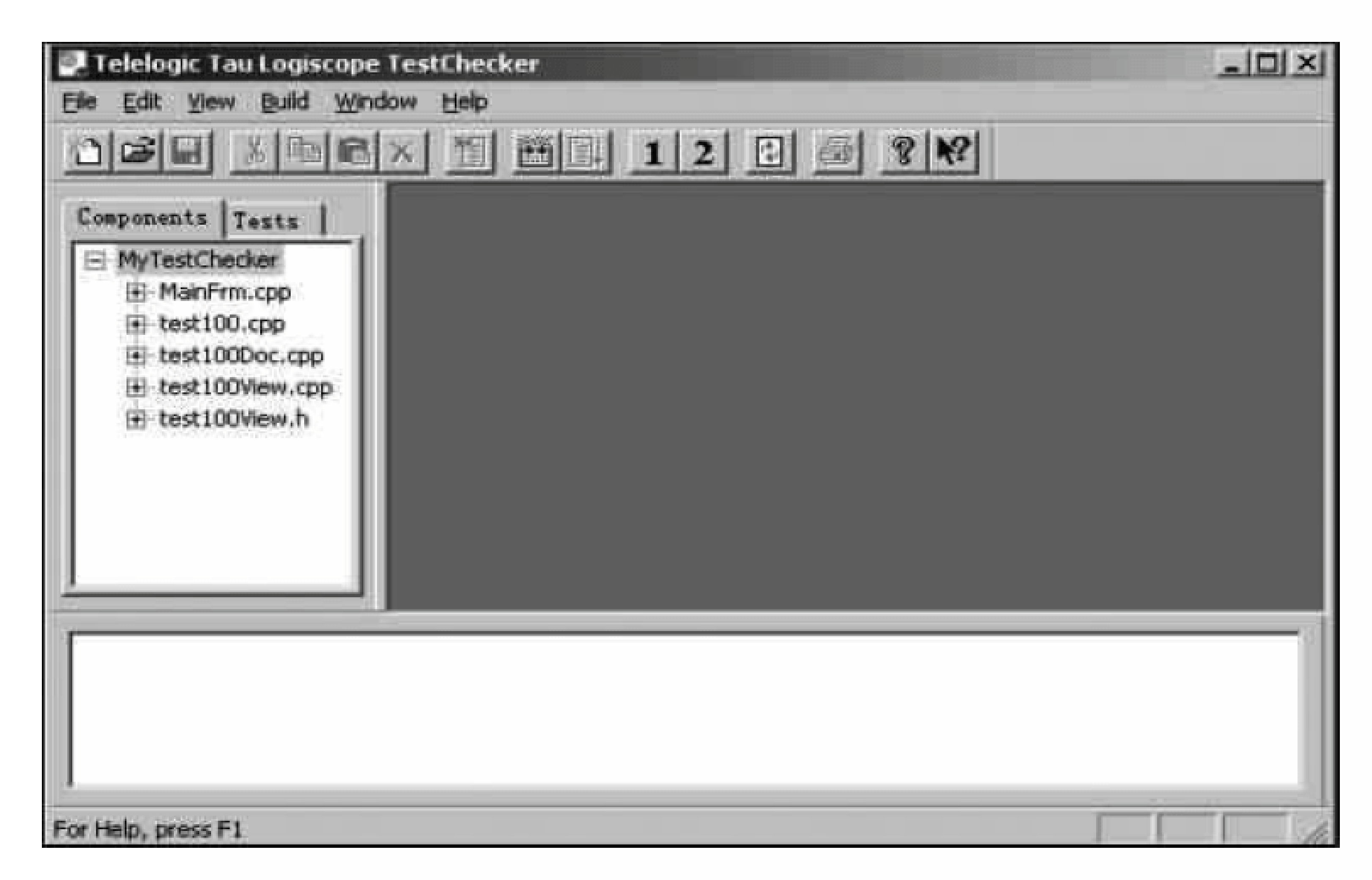


图 16.54 TestChecker 界面

图 16.55 展示了 TestChecker 工具条中几个重要的按钮。

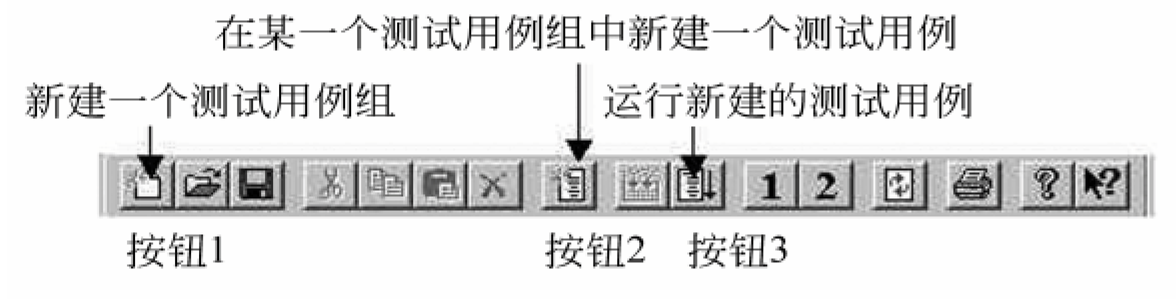


图 16.55 TestChecker 工具条

操作这三个按钮可以建立、运行测试用例：

- 单击按钮 1，会新建一个测试用例组。一个测试用例组可容纳多个测试用例。
- 单击按钮 2，会在一个测试用例组中新建一个测试用例。
- 单击按钮 3，运行新建的测试用例。

按照测试用例事先制定好的操作步骤，执行测试用例。在执行完测试用例，退出被测试的程序后，TestChecker 会给出执行该测试用例后，程序的覆盖情况，如图 16.56 所示。

在树状视图中，双击某一个测试用例，会显示运行该测试后各个函数的覆盖情况，如图 16.57 所示。

选择 View→DDP SPY 命令，显示到目前为止总的覆盖率，即所有测试用例的覆盖率之和，如图 16.58 所示。

如果几个人共同测试一个应用程序的不同部分，那么，这几个人可以分别在自己的机器上建立 TestChecker 项目，独立运行自己的测试用例，并将覆盖率的结果保存成文件。最后，这几个人的测试用例可以合并到一处，得出应用程序总的测试覆盖率。

最后，保存所有操作，退出 TestChecker，出现图 16.59 所示的对话框，询问是否加载最新的 TestChecker 项目文件，单击“是”按钮。在 Logiscope Studio 中选择 Project→Start Viewer 命令，启动 Viewer，如图 16.60 所示。

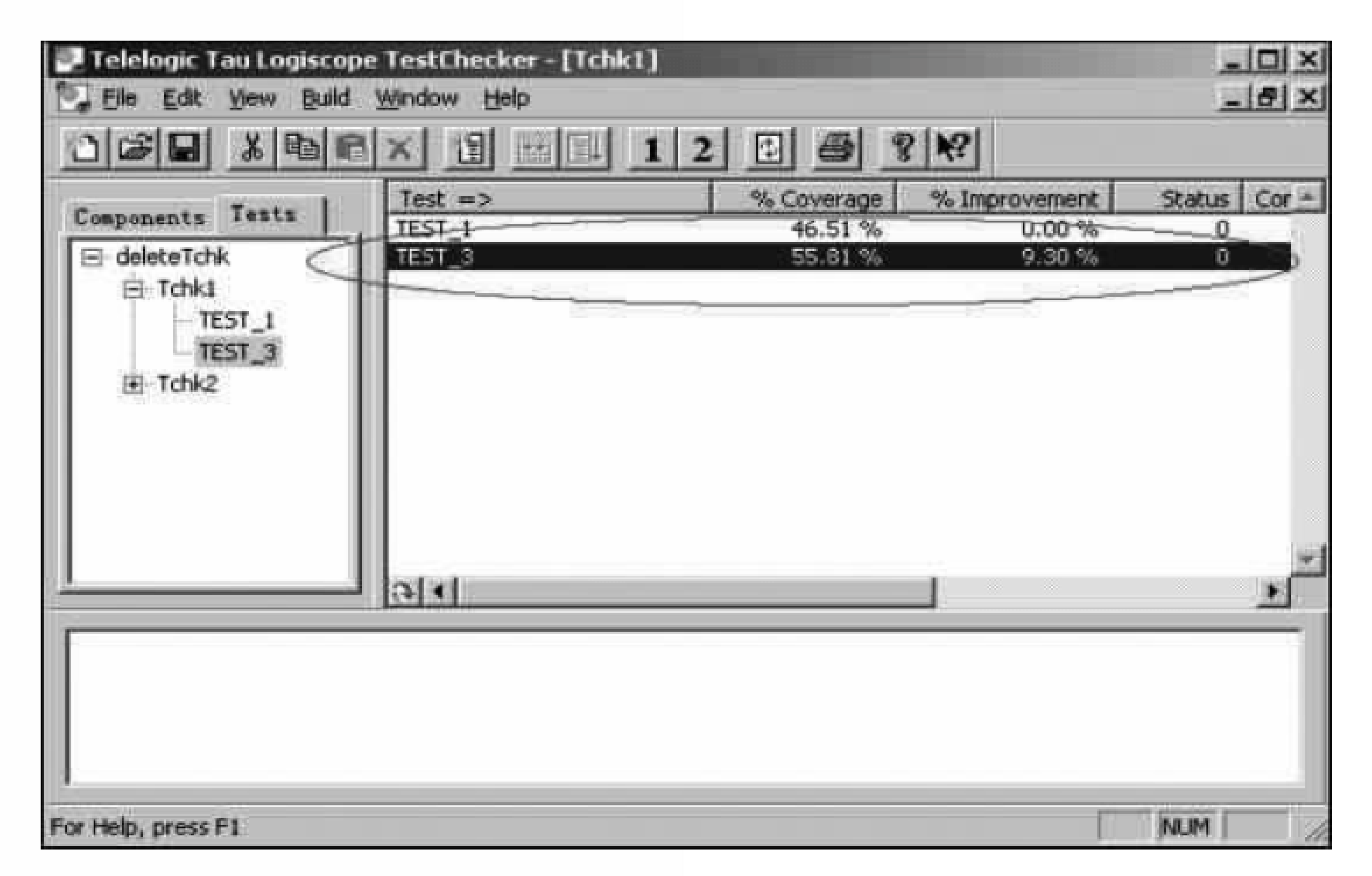


图 16.56　覆盖率情况

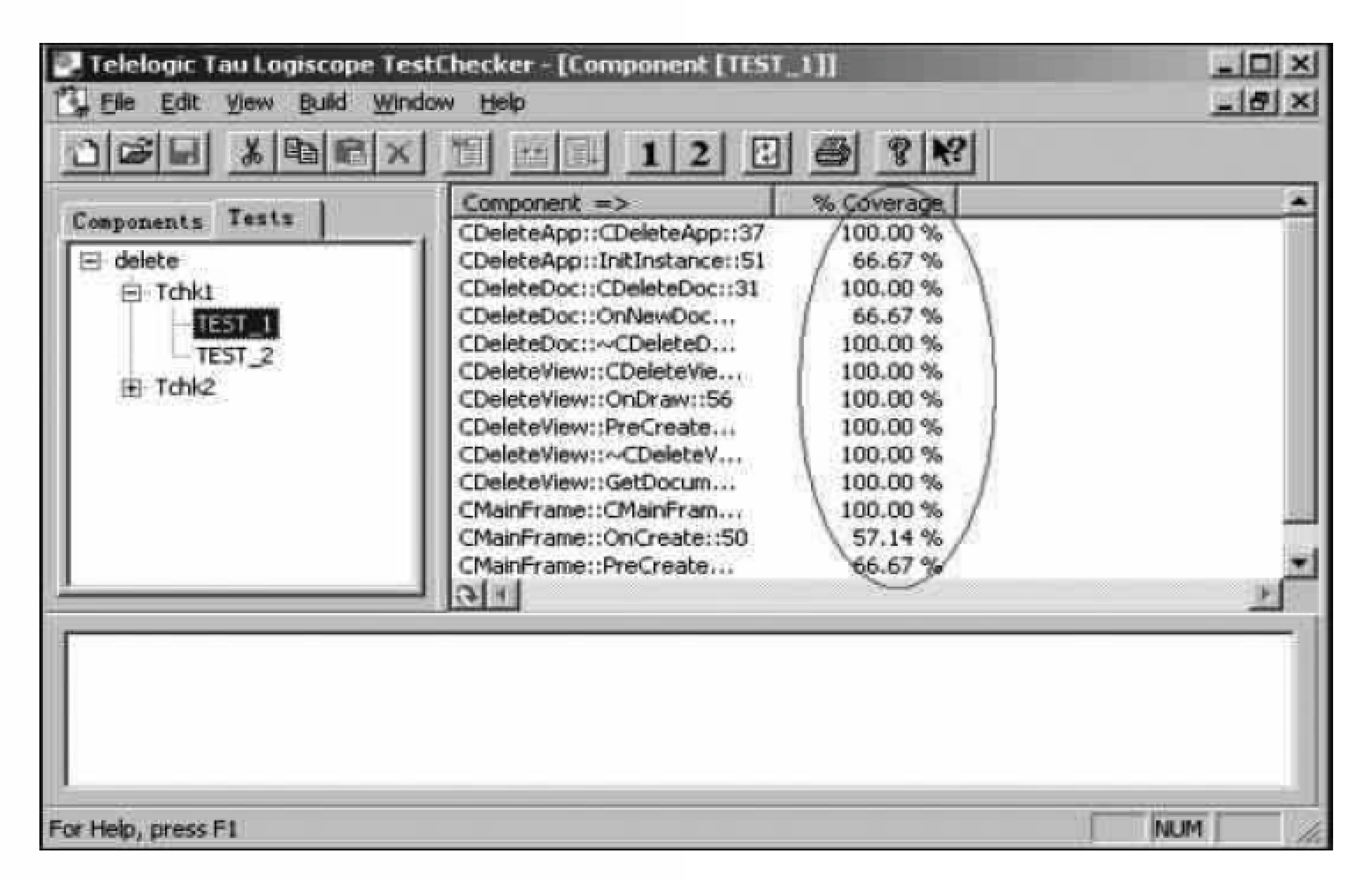


图 16.57　函数覆盖情况

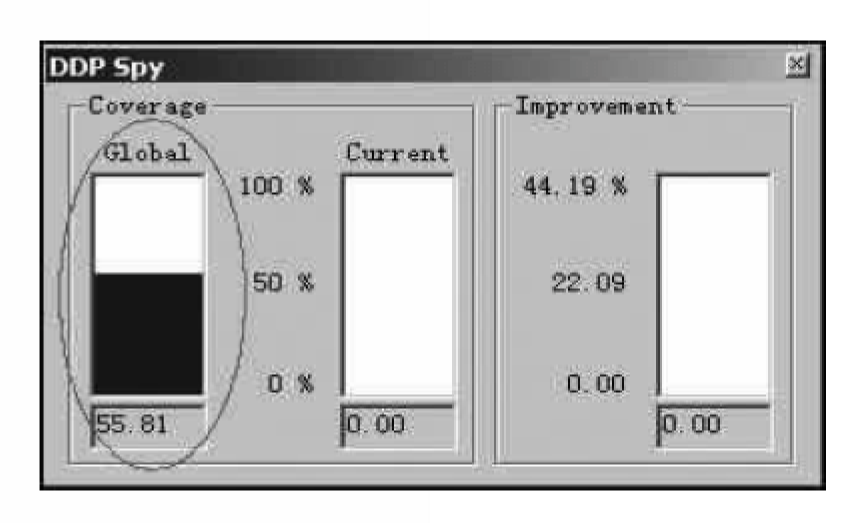


图 16.58　总体覆盖情况



图 16.59　是否重新加载 TestChecker 项目文件

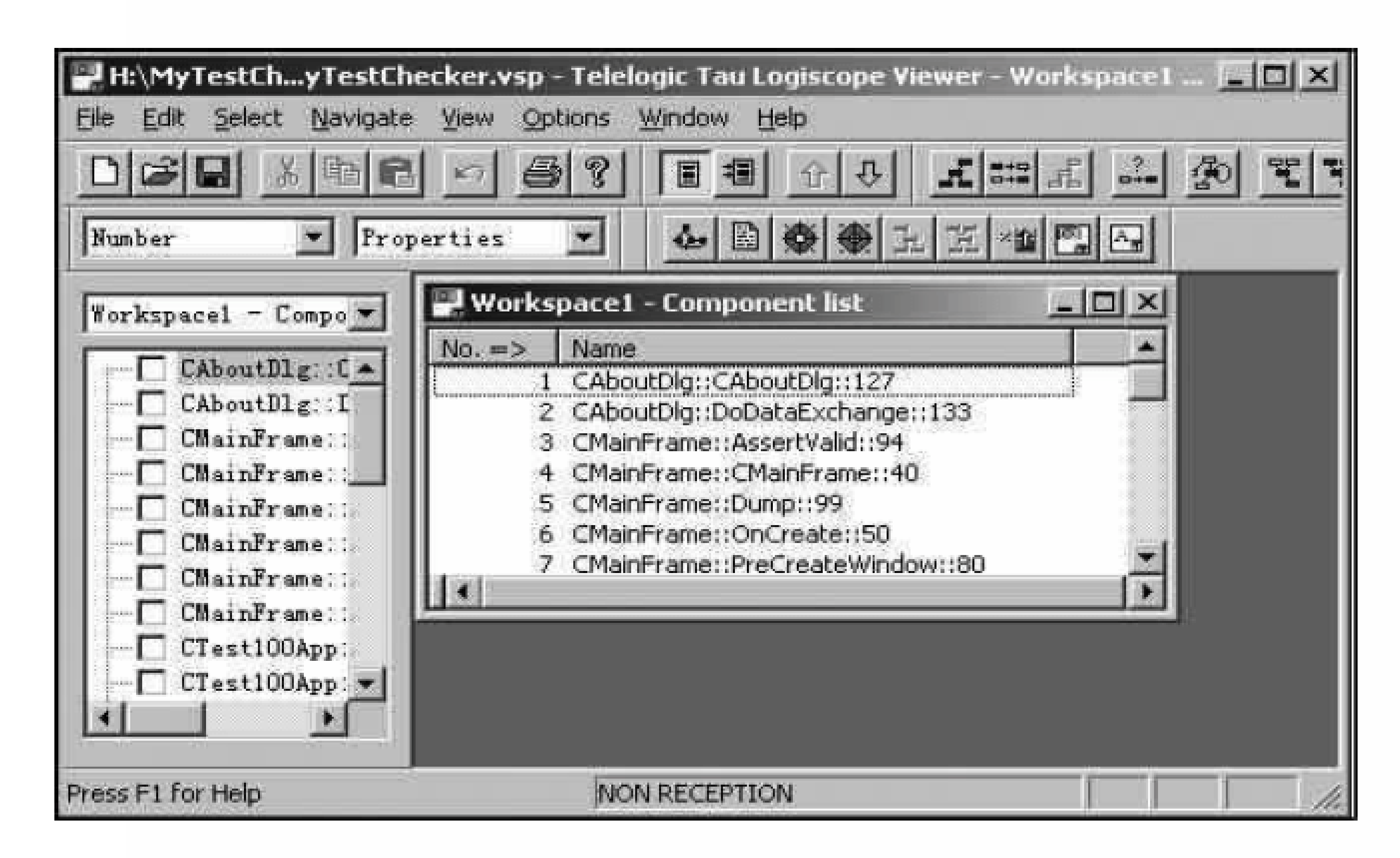


图 16.60　Viewer 界面

在列表框中选择一个函数，在图 16.61 的工具条中，单击按钮 1，会显示当前选中函数的流程图。



图 16.61　Viewer 工具条

再选中 Options→DDP Numbers、Options→Coverage 这两个命令，在函数流程图中会显示目前该函数的覆盖情况，如图 16.62 所示。

其中，实线边代表已被测试覆盖过的路径，虚线边代表还未被测试执行到的路径，数字是不同判断边的编号。

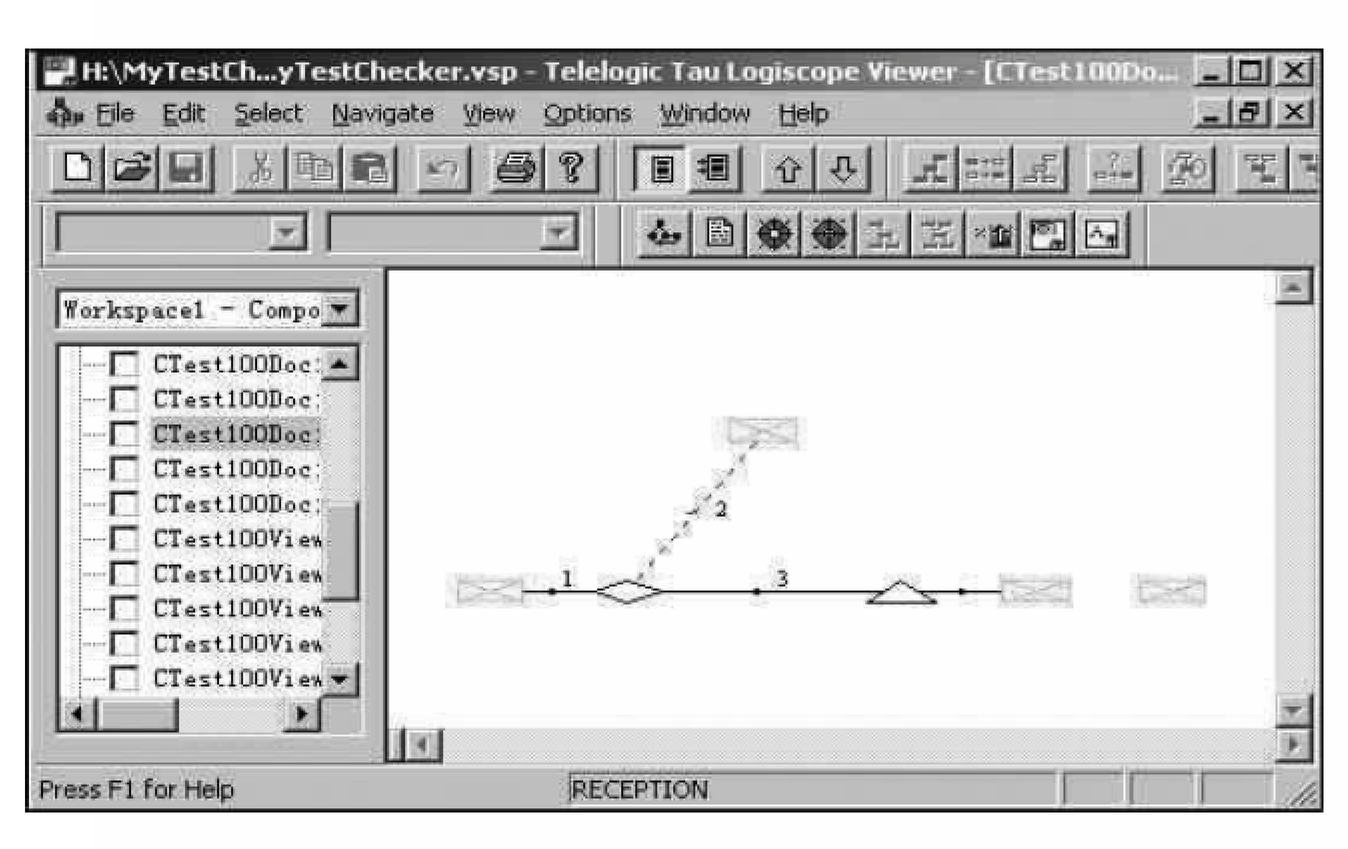


图 16.62　以流程图形式显示覆盖情况

单击图 16.61 工具条上的按钮 2，会显示图 16.63 的数据。

CMainFrame::PreCreateWindow::80 - DDP Coverage

DDP	1	2	3	Rate
TEST_1	1	0	1	66.67
TEST_2	1	0	1	66.67
Total	2	0	2	66.67

图 16.63　以文本形式显示覆盖情况

其中，第一列显示的是不同测试用例的名字，最后一列显示的是执行该测试用例后，函数达到的覆盖率。

在 Logiscope Studio 中，选择 Browse→Test→Test Report 命令，生成网页风格的测试覆盖率统计报告，如图 16.64 所示。报告主要分三部分：第一部分将系统中所有的函数，按其覆盖率的多少，划分成不同的分组；第二部分，列出了每一个函数的覆盖率的详细信息；第三部分给出了所有源文件的清单。

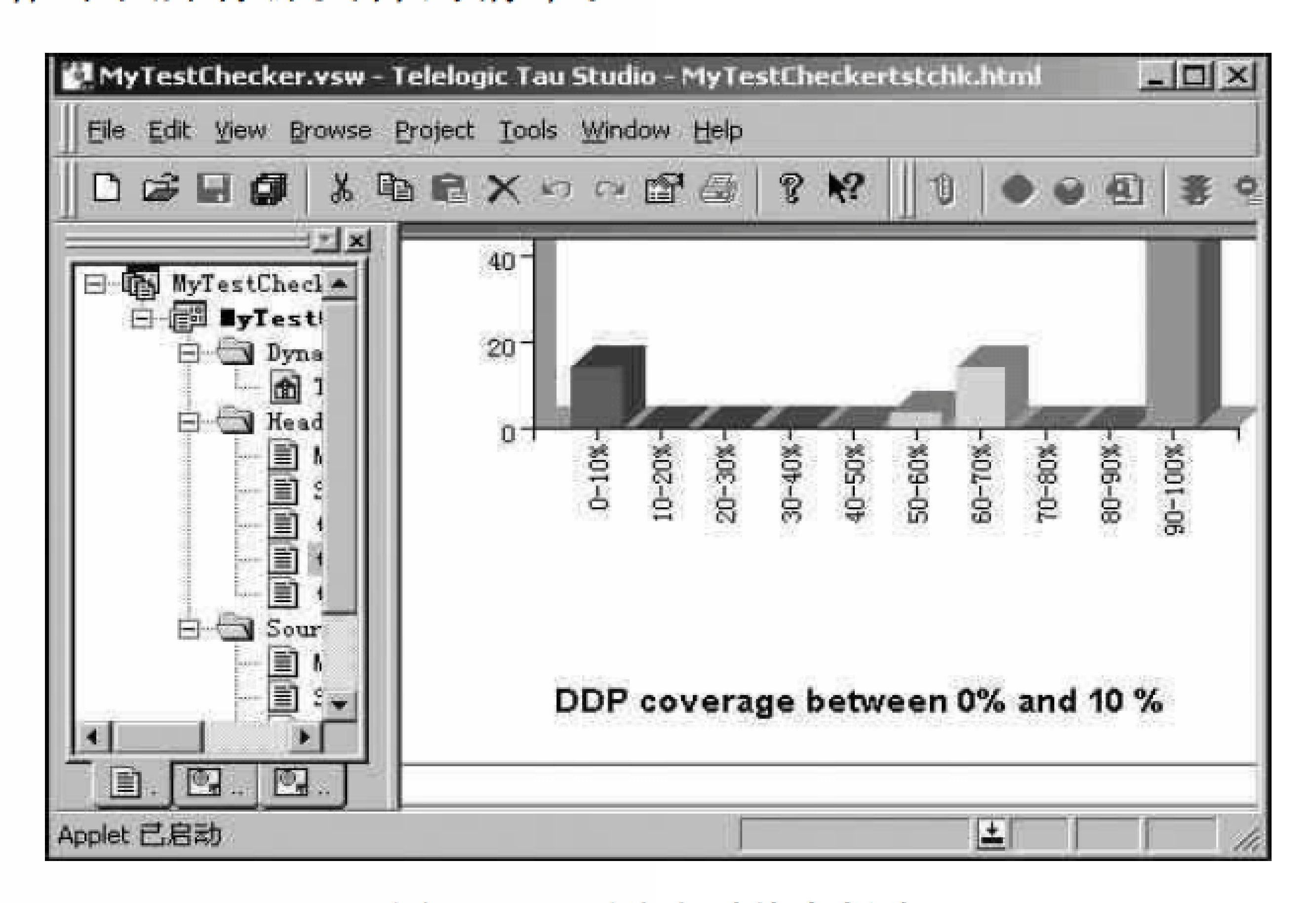
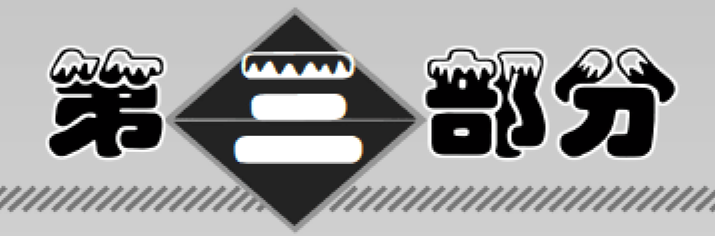
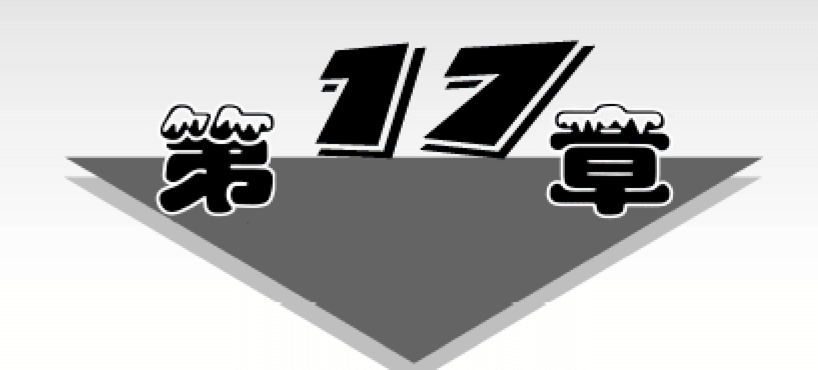


图 16.64　测试覆盖率报告



测试考试指导



全国计算机等级考试四级软件测试工程师

近年来,我国大力设立或引进了各类计算机考试。例如,由国家人事部和原信息产业部组织的“中国计算机软件专业技术资格和水平考试”、国家劳动部组织的“全国计算机信息高新技术考试”、国家教委组织的“全国计算机等级考试”、劳动部职业技能鉴定中心举办的“国家级 Internet 证书培训考试”、国家教委从剑桥大学引入的“剑桥信息技术(CIT)证书考试”、“全国信息应用技术证书(NIT)考试”等,此外,还有“Novell(网络师)认证考试”、各地高校举办的高校计算机等级考试等。另一类为厂方认证,如国际 IT 厂方认证的 Sun Java 认证证书(Sun)-Java、微软认证证书(Microsoft)——MCSE MCP、MCDBA、MCSE、思科认证证书(Cisco)——CCNP 思科认证证书(Cisco)——CCNA、AutoCAD 工程师认证证书等。

计算机技术与软件技术专业技术资格(水平)考试涉及计算机专业的每门课程,还要有数学、外语、系统工程、信息化和知识产权等方面的知识,且注重考查新技术和新方法的应用。考试不但注重广度,而且还有一定的深度。从 2005 年上半年开始,计算机技术与软件技术专业技术资格(水平)考试中增加了软件评测师的考试,旨在培养软件评测师,为我国的软件评测提供专业人才。

17.1 内容介绍

全国计算机等级考试四级软件测试工程师或简称四级软件测试工程师是全国计算机等级考试中四级的一类,属于计算机技术与软件专业资格(水平)考试的中级。

软件测试工程师考试基本要求如下:

(1) 熟悉软件质量、软件测试及软件质量保证的基础知识。

(2) 掌握代码检查、走查与评审的基本方法和技术。

(3) 掌握白盒测试和黑盒测试的测试用例的设计原则和方法。

(4) 掌握单元测试和集成测试的基本策略和方法。

(5) 了解系统测试、性能测试和可靠性测试的基本概念和方法。

(6) 了解面向对象软件和 Web 应用软件测试的基本概念和方法。

(7) 掌握软件测试过程管理的基本知识和管理方法。

(8) 熟悉软件测试的标准和文档。

(9) 掌握QESuite软件测试过程管理平台和QESat/C++软件分析和工具的使用方法。

17.1.1 考试说明

1. 考试要求:

(1) 熟悉计算机基础知识。

(2) 熟悉操作系统、数据库、中间件、程序设计语言基础知识。

(3) 熟悉计算机网络基础知识。

(4) 熟悉软件工程知识,理解软件开发方法及过程。

(5) 熟悉软件质量及软件质量管理基础知识。

(6) 熟悉软件测试标准。

(7) 掌握软件测试技术及方法。

(8) 掌握软件测试项目管理知识。

(9) 掌握C语言及C++或Java语言程序设计技术。

(10) 了解信息化及信息安全基础知识。

(11) 熟悉知识产权相关法律、法规。

(12) 正确阅读并理解相关领域的英文资料。

2. 通过本考试的合格人员能在掌握软件工程与软件测试知识基础上,运用软件测试管理办法、软件测试策略、软件测试技术,独立承担软件测试项目;具有工程师的实际工作能力和业务水平。

3. 本考试设置的科目包括:

(1) 软件工程与软件测试基础知识,考试时间为150分钟,笔试,选择题;

(2) 软件测试应用技术,考试时间为150分钟,笔试,问答题。

17.1.2 考试大纲及考试重点

考试科目1 软件工程与软件测试基础知识

1 计算机系统基础知识

1.1 计算机系统构成及硬件基础知识

- 计算机系统的构成;
- 处理机;
- 基本输入输出设备;
- 存储系统。

1.2 操作系统基础知识

- 操作系统的中断控制、进程管理、线程管理;
- 处理机管理、存储管理、设备管理、文件管理、作业管理;
- 网络操作系统和嵌入式操作系统基础知识;
- 操作系统的配置。

1.3 数据库基础知识
- 数据库基本原理；
- 数据库管理系统的功能和特征；
- 数据库语言与编程。

1.4 中间件基础知识

1.5 计算机网络基础知识
- 网络分类、体系结构与网络协议；
- 常用网络设备；
- Internet 基础知识及其应用；
- 网络管理。

1.6 程序设计语言知识
- 汇编、编译、解释系统的基础知识；
- 程序设计语言的基本成分(数据、运算、控制和传输、过程(函数)调用)；
- 面向对象程序设计；
- 各类程序设计语言的主要特点和适用情况；
- C 语言以及 C++（或 Java)语言程序设计基础知识。

2 标准化基础知识
- 标准化的概念(标准化的意义、标准化的发展、标准化机构)；
- 标准的层次(国际标准、国家标准、行业标准、企业标准)；
- 标准的类别及生命周期。

3 信息安全知识
- 信息安全基本概念；
- 计算机病毒及防范；
- 网络入侵手段及防范；
- 加密与解密机制。

4 信息化基础知识
- 信息化相关概念；
- 与知识产权相关的法律、法规；
- 信息网络系统、信息应用系统、信息资源系统基础知识。

5 软件工程知识

5.1 软件工程基础
- 软件工程概念；
- 需求分析；
- 软件系统设计；
- 软件组件设计；
- 软件编码；
- 软件测试；
- 软件维护。

5.2 软件开发方法及过程

- 结构化开发方法；
- 面向对象开发方法；
- 瀑布模型；
- 快速原型模型；
- 螺旋模型。

5.3 软件质量管理

- 软件质量及软件质量管理概念；
- 软件质量管理体系；
- 软件质量管理的目标、内容、方法和技术。

5.4 软件过程管理

- 软件过程管理概念；
- 软件过程改进；
- 软件能力成熟度模型。

5.5 软件配置管理

- 软件配置管理的意义；
- 软件配置管理的过程、方法和技术。

5.6 软件开发风险基础知识

- 风险管理；
- 风险防范及应对。

5.7 软件工程有关的标准

- 软件工程术语；
- 计算机软件开发规范；
- 计算机软件产品开发文件编制指南；
- 计算机软件需求规范说明编制指南；
- 计算机软件测试文件编制规范；
- 计算机软件配置管理计划规范；
- 计算机软件质量保证计划规范；
- 数据流图、程序流程图、系统流程图、程序网络图和系统资源图的文件编制符号及约定。

6 软件评测师职业素质要求

- 软件评测师职业特点与岗位职责；
- 软件评测师行为准则与职业道德要求；
- 软件评测师的能力要求。

7 软件评测知识

7.1 软件测试基本概念

- 软件质量与软件测试；
- 软件测试定义；

- 软件测试目的；
- 软件测试原则；
- 软件测试对象。

7.2 软件测试过程模型

- V 模型；
- W 模型；
- H 模型；
- 测试模型的使用。

7.3 软件测试类型

- 单元测试、集成测试、系统测试；
- 确认测试、验收测试；
- 开发方测试、用户测试、第三方测试；
- 动态测试、静态测试；
- 白盒测试、黑盒测试、灰盒测试。

7.4 软件问题分类

- 软件错误；
- 软件缺陷；
- 软件故障；
- 软件失效。

7.5 测试标准

7.5.1 GB/T 16260.1—2003 软件工程 产品质量 第1部分：质量模型；

7.5.2 GB/T 18905.1—2002 软件工程 产品评价 第1部分：概述；

7.5.3 GB/T 18905.5—2002 软件工程 产品评价 第5部分：评价者用的过程。

8 软件评测现状与发展

- 国内外现状；
- 软件评测发展趋势。

9 专业英语

- 正确阅读并理解相关领域的英文资料。

考试科目2 软件测试应用技术

1 软件生命周期测试策略

1.1 设计阶段的评审

- 需求评审；
- 设计评审；
- 测试计划与设计。

1.2 开发与运行阶段的测试

- 单元测试；
- 集成测试；

- 系统(确认)测试;
- 验收测试。

2　测试用例设计方法

2.1　白盒测试设计

- 白盒测试基本技术;
- 白盒测试方法。

2.2　黑盒测试用例设计

- 测试用例设计方法;
- 测试用例的编写。

2.3　面向对象测试用例设计

2.4　测试方法选择的策略

- 黑盒测试方法选择策略;
- 白盒测试方法选择策略;
- 面向对象软件的测试策略。

3　软件测试技术与应用

3.1　软件自动化测试

- 软件自动化测试基本概念;
- 选择自动化测试工具;
- 功能自动化测试;
- 负载压力自动化测试。

3.2　面向对象软件的测试

- 面向对象测试模型;
- 面向对象分析的测试;
- 面向对象设计的测试;
- 面向对象编程的测试;
- 面向对象的单元测试;
- 面向对象的集成测试;
- 面向对象的系统测试。

3.3　负载压力测试

- 负载压力测试基本概念;
- 负载压力测试解决方案;
- 负载压力测试指标分析;
- 负载压力测试实施。

3.4　Web 应用测试

- Web 应用的测试策略;
- Web 应用设计测试;
- Web 应用开发测试;
- Web 应用运行测试。

3.5 网络测试
- 网络系统全生命周期测试策略；
- 网络仿真技术；
- 网络性能测试；
- 网络应用测试。

3.6 安全测试
- 测试内容；
- 测试策略；
- 测试方法。

3.7 兼容性测试
- 硬件兼容性测试；
- 软件兼容性测试；
- 数据兼容性测试；
- 新旧系统数据迁移测试；
- 平台软件测试。

3.8 易用性测试
- 功能易用性测试；
- 用户界面测试。

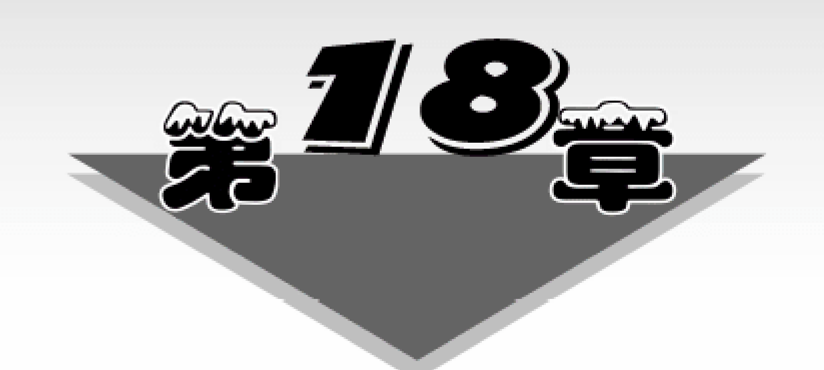
3.9 文档测试
- 文档测试的范围；
- 用户文档的内容；
- 用户文档测试的要点；
- 用户手册的测试；
- 在线帮助的测试。

4 测试项目管理
- 测试过程的特性与要求；
- 软件测试与配置管理；
- 测试的组织与人员；
- 测试文档；
- 软件测试风险分析；
- 软件测试的成本管理。

17.2 相关资料

[1] 张友生. 软件评测师考试考点分析与真题详解[M]. 北京：电子工业出版社，2005.
[2] 教育部考试中心. 全国计算机等级考试四级教程：软件测试工程师(2008年版)[M]. 北京：高等教育出版社，2007.
[3] 王健，苗勇，刘郢. 软件测试员培训教材[M]. 北京：电子工业出版社，2003.

[4] 蔡为东.软件测试工程师面试指导[M].北京:科学出版社,2007.
[5] 全国计算机等级考试[EB/OL].[2013-5-20].http://www.ncre.cn/ncre_new.
[6] 全国计算机等级考试官方论坛[EB/OL].[2013-5-20].http://bbs.ncre.cn.
[7] 全国计算机等级考试网:无忧考试吧[EB/OL].[2013-5-20].http://www.wyks8.com/ncre.
[8] 全国计算机等级考试(NCRE)网[EB/OL].[2013-5-20].http://www.51test.net/ncre.
[9] 考试吧-计算机等级考试第一门户[EB/OL].[2013-5-12].http://www.exam8.com/computer/djks.
[10] 全国计算机软考软件评测师考试[EB/OL].[2013-5-12].http://www.examda.com/soft/zhongji/pingce.



软件测试行业

18.1 测试行业现状

据资料显示，IE4.0 的代码开发时间为 6 个月，而测试用了 8 个月的时间。开发 Windows 2000 操作系统用时 3 年，投入 50 亿美元，使用了 250 名项目经理、1700 名软件开发工程师、3200 名软件测试工程师。当前软件行业比较发达的国家与地区，如欧美、印度、以色列等，软件测试行业的产值几乎占了软件行业总产值的 1/4。软件测试已经成为一个独立的产业，软件测试工程师和开发工程师的比例基本维持在 1∶1 左右，即1 个软件开发工程师便需要有 1 个软件测试工程师。测试行业有如下特点。

(1) 软件测试在软件公司中占有重要地位，其软件测试在人员配备和资金投入方面占据相当的比重，从投入的资金和人力物力来看，测试、使产品稳定和修改花去的时间占到整个项目时长的 80%。

(2) 软件测试理论研究蓬勃发展，引领软件测试理论研究的国际潮流。

(3) 软件测试市场繁荣，如 MI、Compuware、Rational 等，其出品的测试工具占领了国际市场。

(4) 软件产品的认定，往往需要第三方测试的介入。

随着中国软件业的日益壮大和逐步走向成熟，软件测试也在不断发展。从最初的由软件编程人员兼职测试到软件公司组建独立专职测试部门。测试工作也从简单测试演变为包括编制测试计划、编写测试用例、准备测试数据、编写测试脚本、实施测试、测试评估等多项内容的正规测试。测试方式则由单纯手工测试发展为手工、自动兼之，并有向第三方专业测试公司发展的趋势。

当前，国内 120 万软件从业人员中，真正能担当软件测试职位的不超过 5 万人，软件测试人才缺口高达 30 万。国内测试仍然停留在开发人员自行测试阶段，软件开发和测试人员结构明显失调，缺乏第三方测试。国内软件测试与国外软件测试主要存在如下差距：

(1) 测试的理解认识。国内软件企业普遍存在重开发轻测试，将测试置于从属地位，没有认识到软件项目完成不仅取决于开发人员，更取决于测试人员。国内许多中小型软件企业没有软件测试部门，有些甚至不设置软件测试的岗位。

(2) 测试过程的管理。国内软件企业普遍存在测试的随意化、简单化,未建立有效、规范的测试管理体系。

(3) 测试工具的使用。国内软件企业的测试普遍缺乏使用自动化测试工具。

(4) 测试人员的培养。软件测试领域目前十分缺乏人才,首先是测试人员角色定位不合理,其次是缺乏专家级测试人才。

当然,国内软件测试产业也正在慢慢发展。第一,软件测试工具的开发取得了显著进展,如西安交通大学开发的 COBOL 测试系统、华中科技大学开发的 C 编译程序测试系统,北京航空航天大学与清华大学开发的 C 软件综合测试系统、中科院开发的 I-test 测试工具等。第二,软件公司企业为软件测试设置了相应部门,如东软、神州数码等软件企业。第三,软件测试行业正在满足越来越多的就业需求。

18.2 软件测试职位

软件测试工程师是指理解产品的功能要求,并对其进行测试,检查软件有没有错误,决定软件是否具有稳定性,写出相应的测试规范和测试用例的专门工作人员。简而言之,软件测试工程师在一家软件企业中担当的是"质量管理"角色,及时纠错及时更正,确保产品的正常运作。软件测试工程师的发展进阶之路有哪些? 微软公司的陈宏刚博士介绍说,"软件测试人员一般有三大发展方向。"一是走软件测试的技术路线,成长为高级软件测试工程师。二是向管理方向发展,从测试工程师到组长,再到测试经理,以至更高的职位。三是转换职业,做项目管理或做开发人员。

1. 技术方向

按其级别和职位的不同,软件测试工程师可分为初级软件测试工程师、中级软件测试工程师、高级软件测试工程师三类。初级软件测试工程师通常都是按照软件测试方案和流程对产品进行功能测验,检查产品是否有缺陷。一般是刚入门测试领域或具有一些手工测试经验的个人。中级软件测试工程师则编写软件测试方案、测试文档,与项目组一起制定软件测试阶段的工作计划,能够在项目运行中合理利用测试工具完成测试任务。一般具有 1～2 年经验的测试工程师或程序员,可以编写自动测试脚本程序并担任测试编程初期领导工作。而高级软件测试工程师则要熟练掌握软件测试与开发技术,且对所测试软件对口行业非常了解,能够对可能出现的问题进行分析评估。一般具有 3～4 年经验的测试工程师或程序员,能帮助开发或维护测试或编程标准与过程,负责同级的评审,并为其他初级的测试工程师或程序员充当顾问。

2. 管理方向

测试人员往管理方面发展,通常有测试负责人和测试经理两种角色。测试负责人是具有 4～6 年经验的测试工程师或程序员,通常负责管理 1～3 名测试工程师或程序员。担负一些进度安排、工作规模/成本估算职责和预算目标交付产品。测试经理往往又被称

为质量保证经理，一般具有 10 多年的工作经验，管理 8 名或更多的人员参加的 1 个或多个项目，负责测试、质量保证领域内的整个开发生存周期业务。

18.3 软件测试思维方式

软件测试人员一般应具备的思维方式有：逆向思维方式、组合思维方式、全局思维方式、两极思维方式、简单思维方式、比较思维方式和发散性思考方式等。

1. 逆向思维方式

逆向思维是相对的，就是按照与常规思路相反的方向进行思考，比如将根据结果逆推条件，从而得出输入条件的等价类划分，其实逆向思维在调试当中用到的也比较多，当发现缺陷时，进一步定位问题的所在，往往就是逆向分析，发现开发人员思维的漏洞。

2. 组合思维方式

当将相关的事物组合在一起却能发现很多问题，如多进程并发，不但使得程序的复杂度提高，也让程序的缺陷率随之增长，针对不同的应用，可以酌情考虑使用“排列”或者“组合”，将相关的因素划分到不同的维度上，然后再考虑其相关性。

3. 全局思维方式

全局思维方式就是从多角度分析待测的系统，以不同角色去看系统，分析其是否能够满足需求。在软件开发过程中，进行的各种评审，让更多的人参与思考，尽可能的实现全方位审查某个解决方案的正确性以及其他特性。

4. 两极思维方式

两极思维方式，是在极端的情况下，看是否存在缺陷？边界值分析方法就是两极思维方式的典范。

5. 比较思维方式

人们认识事物时，往往都是和已有的某些概念进行比较，找出相同、相异之处，或者归类。应用模式是“比较思维”很常见的例子，有设计模式、体系结构模式等，由于经验在测试中很重要，比较思维是较为常用的方式。

6. 发散性思考方式

发散性思考其实就是一种寻求多种答案，最终使问题获得解决的思考方法。例如，某嵌入式软件 U 盘导出数据的功能，如表 18.1 所示。

表 18.1 U 盘导出数据的功能

考虑方向	检查点	测试思路简述	备注
正向功能	导出数据的正确性	用 U 盘导出某软件的数据，验证其导出数据的正确性	
正向功能	导出功能的有效性处理	某软件待导出数据不存在时，导出功能是否正确	
逆向功能	导出功能的配置	软件安装后，导出功能的配置是否正确	此点与软件实现功能有关
边界容量	U 盘空间不足时的处理	当前 U 盘空间只能容纳待导出的部分数据时，执行数据导出	
边界容量	U 盘空间满时的处理	U 盘空间满时，执行数据导出	
容错	U 盘写保护处理	U 盘写保护时，执行数据导出	
容错	坏 U 盘的处理	用一个坏 U 盘执行数据导出	坏 U 盘的准入条件可以 Windows 能否正常写入数据为准
容错	人为非法操作容错处理	导出数据过程中，拔出 U 盘，软件的后续处理是否有异常	
容错	软件工作时，遇特殊情况的容错处理	数据导出过程中，突然断电，再开机检查 U 盘中的数据及软件导出功能是否符合预期	

18.4 常用软件测试工程师笔试题

（1）软件测试的目的正确的是（　　）。

① 测试是为了发现程序中的错误而执行程序的过程

② 好的测试方案是极可能发现迄今为止尚未发现的错误的测试方案

③ 成功的测试是发现了至今为止尚未发现的错误的测试

④ 测试并不仅仅是为了找出错误. 通过分析错误产生的原因和错误的发生趋势，可以帮助项目管理者发现当前软件开发过程中的缺陷，以便及时改进

A. ①　　B. ①②③　　C. ②③④　　D. ①②③④

（2）软件测试的对象包括（　　）。

A. 目标程序和相关文档

B. 源程序、目标程序、数据及相关文档

C. 目标程序、操作系统和平台软件

D. 源程序和目标程序

（3）从软件内部结构和具体实现的角度划分软件测试种类，软件测试分为（　　）。

A. 静态测试、动态测试

B. 黑盒测试、白盒测试、灰盒测试

C. 单元测试、集成测试、确认测试、系统测试、验收测试

D. 以上都不对

(4) 关于软件测试模型,描述正确的是(　　)。

A. V 模型测试的对象就是程序本身,测试与开发可以同一阶段进行

B. W 模型测试的对象是程序,需求、设计等,可以支持迭代的开发模型

C. H 模型软件测试过程活动完全独立,贯穿产品整个生命周期,与其他流程并发地进行

D. X 模型是事先计划再进行测试

(5) 软件测试按实施组织分,测试应该包括以下的(　　)。

① 开发方测试　② 用户方测试　③ 第三方测试　④ 验收测试　⑤ 确认测试

A. ①②③　　B. ③④⑤　　C. ①②④　　D. ①②③④⑤

(6) 测试计划的步骤为(　　)。

A. 确定项目管理机制,预计测试工作量,测试计划评审

B. 确定测试范围,确定测试策略,确定测试标准,预计测试工作量

C. 确定测试构架,确定项目管理机制,预计测试工作量,测试计划评审

D. 确定测试范围,确定测试策略,确定测试标准,确定测试构架,确定项目管理机制,预计测试工作量,测试计划评审

(7) 为保证测试活动的可控性,必须在软件测试过程中进行软件测试配置管理,一般来说,软件测试配置管理中最基本的活动包括(　　)。

A. 配置项标识、配置项控制、配置状态报告、配置审计

B. 配置基线确立、配置项控制、配置报告、配置审计

C. 配置项标识、配置项变更、配置审计、配置跟踪

D. 配置项标识、配置项控制、配置状态报告、配置跟踪

(8) 某次程序调试没有出现预计的结果,下列选项中(　　)不可能是导致出错的原因。

A. 变量没有初始化　　B. 编写的语句书写格式不规范

C. 循环控制出错　　D. 代码输入有误

(9) 下列关于程序效率的描述错误的是(　　)。

A. 提高程序的执行速度可以提高程序的效率

B. 降低程序占用的存储空间可以提高程序的效率

C. 源程序的效率与详细设计阶段确定的算法的效率无关

D. 好的程序设计可以提高效率

(10) 程序设计语言中(　　)。

A. while 循环语句的执行效率比 do-while 循环语句的执行效率高

B. while 循环语句的循环体执行次数比循环条件的判断次数多 1,而 do-while 语句的循环体执行次数比循环条件的判断次数少 1

C. while 语句的循环体执行次数比循环条件的判断次数少 1,而 do-while 语句的循环体执行次数比循环条件的判断次数多 1

D. while 语句的循环体执行次数比循环条件的判断次数少 1,而 do-while 语句的循环体执行次数等于循环条件的判断次数

(11) DB、DBMS 和 DBS 三者间的关系是(　　)。

A. DB 包括 DBMS 和 DBS　　B. DBS 包括 DB 和 DBMS

C. DBMS 包括 DBS 和 DB　　D. DBS 与 DB、DBMS 无关

(12) 在 TCP/IP 模型中,应用层包含了所有的高层协议,在下列的一些应用协议中,(　　)是能够实现本地与远程主机之间的文件传输工作。

A. Telnet　　B. FTP　　C. SNMP　　D. NFS

(13) 一个局域网中某台主机的 IP 地址为 176.68.160.12,使用 22 位作为网络地址,那么该局域网的子网掩码和最多可以连接的主机数分别为(　　)。

A. 255.255.251.0 和 1021　　B. 255.255.252.0 和 1022

C. 255.255.253.0 和 1023　　D. 255.255.254.0 和 1024

(14) Linux 文件系统的文件都按其作用分门别类地放在相关的目录中,对于外部设备文件,一般应将其放在(　　)目录中。

A. /bin　　B. /etc　　C. /dev　　D. /lib

(15) 某公司采用的软件开发过程通过了 CMM2 认证,表明该公司(　　)。

A. 开发项目成效不稳定,管理混乱

B. 对软件过程和产品质量建立了定量的质量目标

C. 建立了基本的项目级管理制度和规程,可对项目的成本、进度进行跟踪和控制

D. 可集中精力采用新技术新方法,优化软件过程

(16) 质量管理人员在安排时间进度时,为了能够从全局出发、抓住关键路径、统筹安排、集中力量,从而达到按时或提前完成计划的目标,可以使用(　　)。

A. 活动网络图　　B. 因果图

C. 优先矩阵图　　D. 检查表

(17) 某公司最近承接了一个大型信息系统项目,项目整体压力较大,对这个项目中的变更,可以使用(　　)等方式提高效率。

① 分优先级处理　② 规范处理　③ 整批处理　④ 分批处理

A. ①②③　　B. ①②④

C. ②③④　　D. ①③④

(18) 下面工作中制订进度计划的基础的是(　　)。

A. 工作分解结构(WBS)

B. 网络图

C. 甘特图(GANTT)

D. 资源平衡(RESOURCE LEVELLING)

(19) 下面方法中通常不会被用来缩短进度的是(　　)。

A. 赶工

B. 变更范围

C. 以并行方式而不是序列方式来实施进行活动

D. 资源平衡

(20) 为了提高测试的效率，通常的做法是(   )。

A. 选择发现错误可能性大的数据作为测试用例

B. 在完成程序的编码之后在指定软件的测试计划

C. 随机选取测试用例

D. 取一切可能的输入数据作为测试用例



微软公司软件测试

微软公司内部的软件测试人员与软件开发人员的比例一般为1.5～2.5左右，即一个开发人员对应至少两位测试人员，从而确保软件产品质量。测试人员分为两类：测试工具软件开发工程师和软件测试工程师。测试工具软件开发工程师主要负责编写测试工具代码，并利用测试工具对软件进行测试；或者开发测试工具为软件测试工程师服务。软件测试工程师主要负责测试软件的功能，检查软件的稳定性，并写出相应的测试规范和测试案例。微软认为，测试人员应站在使用者的角度上，不断地使用和"攻击"开发的软件产品，尽量多地找出产品中存在的问题。

19.1 微软测试策略

第2章讲到了软件测试的几种观点，其中讲到了软件测试的辩证论，微软的测试策略是将正向思维和反向思维这两类测试方法结合起来，以正向思维为基础和主要线索，阶段性地运用反向思维方法。

1. 正向思维测试

微软的正向思维测试总体上说分为三个步骤进行：一是审核需求和设计，二是设计测试，三是实施运行测试。

(1) 审核需求和设计，以需求和设计为本来验证软件的正确性。一般包括：

① 由项目经理根据用户需求编写的需求文本。

② 由项目经理根据需求文本而编写的功能设计文本。

③ 由开发人员根据功能文本而编写的实施设计文本。微软的测试人员要参与所有这些文本的审核。作为测试人员，审核重点是检查文本对用户需求定义的完整性、严密性和功能设计的可测性。

(2) 测试人员要根据已审核通过的需求和设计编制测试计划，设计测试用例。从用户角度进行的黑盒测试，实现"测试计划"和"测试用例设计"两个文本。"测试计划" 文本主要阐述测试的范畴、领域、方法、工具、资源和计划时间表等等。"测试用例设计"文本要列出测试用例、每个用例的设置、执行步骤和预期结果。

(3) 实施运行测试。这是整个开发过程中最长最复杂的一个阶段。此阶段的测试在周密的计划下进行，将上一步设计的测试用例按计划付诸实施的过程，根据产品的架构和

功能模块的依赖关系，按照项目的总体计划共同推进。从测试的过程来看，总是先运行或执行简单用例，然后再复杂用例；先验证单一的基本功能，再综合的端到端的功能；先发现解决表面的，影响面大的缺陷，再深层的，不容易重现的缺陷。

2. 反向思维测试

微软的反向思维测试是阶段性的，常常根据需要而带有随机性和突击性。对于这类测试，在微软有一个专门的名称：Bug Bash（译为：缺陷大扫除）。缺陷大扫除通常发生在项目开发各阶段的末期，如 Beta 版发布前的一个专门的时间段，在这期间所有参与项目的人员，集中全部精力，运用各方面的知识搜寻项目的缺陷。一般有以下要点：

（1）尽管这是一个测试活动，但参与者并不仅限于测试人员。项目经理，开发人员甚至于高层管理人员都应参加，集思广益。

（2）要鼓励各部门，领域交叉搜索，因为新的思路和视角通常有助于发现更多的缺陷。

（3）为调动积极性，引入竞争机制，比如当活动结束时，评出发现缺陷最多，发现最严重缺陷的个人，给以物质和精神奖励。

（4）往往以测试专题展开，如安全性、用户界面可用性、国际化和本地化等等。

微软的第二类测试除了缺陷大扫除外，经常还有一些专业性的测试，最典型的是针对安全性攻击测试。一般会邀请公司内部，或业界的专家来搜寻产品的安全漏洞。

微软在测试时主要考虑以下几个问题：

（1）测试要考虑到所有的出错可能性，要做一些不是常规做的事。

（2）除了漏洞之外，测试还应考虑性能问题，保证软件运行良好。

（3）测试要考虑软件的兼容性。

微软测试中使用的测试文档主要包括以下几种：

（1）测试计划。测试计划和产品开发紧密相关，由多个部分组成。所有大型的商业软件都需要完整的测试计划，需要具体到每一个步骤，并且每一个部分都要符合规范要求。

（2）测试规范。测试规范是指为每一个在测试计划中确定的产品领域所写的文档，用来描述该领域的测试需求。编写测试规范，需要参照项目经理写的产品规范，开发人员写的开发计划。每个领域都应该有一份详细的测试规范，所以还需要参照测试计划。

（3）测试案例。测试案例是指描述如何测试某一个领域的文档，这些文档符合测试规范中的需求说明。根据测试规范的测试设定开发，根据测试反馈信息，对于没有考虑到的新问题，不断添加测试案例。测试案例没有固定格式，只要清楚表明了测试步骤和需要验证的事实，使得任何一位测试人员都可以根据测试案例的描述完成测试。

（4）测试报告。测试管理人员以测试报告的形式向整个产品开发部门报告测试结果及发现的缺陷或错误。撰写测试报告的目的是为了让整个产品开发部门了解产品开发的进展情况，以使缺陷或错误能够迅速得到修复。测试报告的格式并无定式，要求能够完整、清楚地反映当前的测试进展情况。

（5）缺陷或错误报告。测试人员以缺陷或错误报告的形式向开发人员报告所发现的

缺陷或错误。撰写缺陷或错误报告的目的是为了使缺陷或错误能够得到修复，测试人员的缺陷或错误报告撰写的好坏会直接影响到开发人员对缺陷或错误的修复。

19.2 一道微软测试题目

考官从办公室（面试现场）随意选取一个简单物品，例如，一个喝水的带广告图案的花纸杯，让应聘人对它设计出尽可能多的测试用例。

微软测试常常考察应试者思维的超常性的角度，这个题目考察应试者的经验、想象力和思维的敏捷性，考官希望得到各种各样的测试用例。应试者可以从“基本功能测试”、“可用性测试”、“安全测试”、“压力测试”、“性能测试”等等角度回答。以下的回答中有不少好的例子，比如“杯子设计是否上大下小，在运输过程中可以套在一起有效利用空间，在使用时也容易拿开”，“为国际化和本地化的需要，广告图案和文字是否在政治、宗教和文化方面具有广泛的适用性”，比如安全性问题。杯子所用的材料（包括纸基、涂层和广告颜料）是否符合食品卫生标准，在内外温度等环境因素下是否会与所盛各种饮料反应，而产生对人体有害的物质。

下面，给出此题目的参考答案：

（1）GUI 测试：

① 看其形状、大小设计是否适合人方便使用。

② 外观是否吸引人，带广告的图案与广告的切合程度。

③ 带广告的图案沾水后是否掉色、模糊。

（2）功能、压力、安全性测试：

① 在杯子内分别装入少量的、半杯的、满杯的热水、冷水、冰水等。

② 看其装载量和装载时间以及纸杯拿在手中的硬度是否达到设计标准。

③ 装入热水后，纸杯是否有异味。

④ 抗摔能力（空杯，半杯水，满杯水，以及不同的温度下）。

⑤ 残疾人士用此杯去喝水的容易程度。

⑥ 检查广告的图案是否容易剥落。

⑦ 24×7 测试：装入液体后记录其多久以后漏水。

参 考 文 献

[1] (美)Chris Wysopal ,Lucas Nelson, Dino Dai Zovi, et,al. 软件安全测试艺术[M]. 程永敬, 译,北京: 机械工业出版社, 2007.
[2] (美)Paul C. Jorgensen. 软件测试(原书第 2 版)[M]. 韩柯, 杜旭涛, 译. 北京: 机械工业出版社,2007.
[3] (美)Srinivasan Desikan, Gopalaswamy Ramesh. 软件测试原理与实践[M]. 韩柯,李娜, 译. 北京: 机械工业出版社, 2007.
[4] (美) 马瑟. 软件测试基础教材(英文版)[M]. 北京: 机械工业出版社,2008.
[5] (美)Daniel J. Mosley Bruce A. Posey. 软件测试自动化[M]. 邓波,黄丽娟,曹青春,等译. 北京: 机械工业出版社,2003.
[6] (美)麦格雷戈. 面向对象的软件测试[M]. 杨文宏, 等译. 北京: 机械工业出版社, 2002.
[7] (美)Glenford J. Myers. 软件测试的艺术(原书第 2 版)[M]. 王峰, 陈杰,译. 北京: 机械工业出版社,2006.
[8] (德)Dirk Huberty. 软件质量和软件测试[M]. 马博, 赵云龙, 译. 北京: 清华大学出版社,2003.
[9] (美)David Gustafson. 软件工程习题与解答[M]. 钟鸣,王君, 等译. 北京: 机械工业出版社,2003.
[10] 段念. 软件性能测试过程详解与案例剖析[M]. 北京: 清华大学出版社,2006.
[11] 赵池龙. 实用软件工程[M]. 北京: 电子工业出版社,2003.
[12] 史九林,古乐. 软件测试技术概论[M]. 北京: 清华大学出版社, 2004.
[13] 董杰. 软件测试精要[M]. 北京: 电子工业出版社出版, 2008.
[14] 贺平. 软件测试教程[M]. 北京: 电子工业出版社, 2006.
[15] 宏刚. 软件开发的科学与艺术[M]. 北京: 电子工业出版社,2002.
[16] 宫云战. 软件测试教程[M]. 北京: 机械工业出版社, 2008.
[17] 路晓丽, 葛玮, 龚晓庆 等. 软件测试技术[M]. 北京: 机械工业出版社, 2007.
[18] 宫云战. 软件测试[M]. 北京: 国防工业出版社, 2006.
[19] 王东刚. 软件测试与 Junit 实践[M]. 北京: 人民邮电出版社,2004.
[20] 王健,苗勇,刘郢. 软件测试员培训教材[M]. 北京: 电子工业出版社,2003.
[21] 朱少民. 软件测试方法和技术[M]. 北京: 清华大学出版社, 2005.
[22] 李幸超. 实用软件测试: 来自硅谷的技术、经验、心得和实例[M]. 北京: 电子工业出版社,2006.
[23] 张友生. 软件评测师考试考点分析与真题详解[M]. 北京: 电子工业出版社,2005.
[24] 郑人杰. 计算机软件测试技术[M]. 北京: 清华大学出版社,1992.
[25] 魏方. 计算机软件测试技术与实例[M]. 北京: 北京希望电脑公司,1991.
[26] 麦格雷戈,杨文宏. 面向对象的软件测试[M]. 北京: 中信出版社,2002.
[27] 朱少民. 全程软件测试[M]. 北京: 电子工业出版社,2007.
[28] 教育部考试中心. 全国计算机等级考试四级教程 : 软件测试工程师(2008 年版)[M]. 北京: 高等教育出版社,2007.
[29] Patton,周予滨,姚静. 软件测试[M]. 北京: 机械工业出版社,2002.
[30] 秦晓. 软件测试[M]. 北京: 科学出版社,2008.
[31] 佟伟光. 软件测试[M]. 北京: 人民邮电出版社,2008.

[32] Gerald D. Everett，Raymond Mcleod Jr，郭耀. 软件测试：跨越整个软件开发生命周期[M]. 北京：清华大学出版社，2008.
[33] 古乐，史九林. 软件测试案例与实践教程[M]. 北京：清华大学出版社，2007.
[34] 蔡为东. 软件测试工程师面试指导[M]. 北京：科学出版社，2007.
[35] 陈能技. 软件测试技术大全：Software Testing GuideFundamentals，Tools and Practice ：测试基础 流行工具 项目实战[M]. 北京：人民邮电出版社，2008.
[36] 陈文滨，朱小梅，任冬梅. 软件测试技术基础[M]. 北京：清华大学出版社，2008.
[37] Mark Fewster，Dorothy Graham，舒智勇. 软件测试自动化技术与实例详解[M]. 北京：电子工业出版社，2000.
[38] 张克东，庄燕滨. 软件工程与软件测试自动化教程[M]. 北京：电子工业出版社，2002.
[39] 林宁，孟庆余. 软件测试实用指南[M]. 北京：清华大学出版社，2004.
[40] 刘怀亮，相洪贵. 软件质量保证与测试[M]. 北京：冶金工业出版社，2007.
[41] 周伟明. 软件测试实践[M]. 北京：电子工业出版社，2008.
[42] 许育诚，王慧文. 软件测试与质量管理[M]. 北京：电子工业出版社，2004.
[43] 赵斌. 软件测试技术经典教程[M]. 北京：科学出版社，2007.
[44] 韩万江. 软件工程案例教程[M]. 北京：机械工业出版社，2009.
[45] 郁莲. 软件测试方法与实践[M]. 北京：清华大学出版社，2008.
[46] 张向宏，陈渌萍. 软件测试理论与实践教程[M]. 北京：人民邮电出版社，2009.
[47] 周元哲. 软件测试教程[M]. 北京：机械工业出版社，2010.
[48] 软件测试[EB/OL]. [2013-03-13]. http://www.uml.org.cn/Test/test.asp.
[49] 51Testing 软件测试网[EB/OL]. [2013-01-10]. http://www.51testing.com/html/index.html.
[50] 软件测试基地[EB/OL]. [2012-12-09]. http://www.cntesting.com.
[51] 软件评测师考试吧[EB/OL]. [2012-11-09]. http://www.exam8.com/computer/spks/rp.
[52] 全国计算机软考软件评测师考试[EB/OL]. [2013-04-09]. http://www.examda.com/soft/zhongji/pingce.
[53] 泽众软件[EB/OL]. [2013-05-02]. http://www.spasvo.com.